Ada Skills Sharpened

Projectlet based approach

2025-02-04

Table of contents

Preface

The year was 1986. I was privileged to be inserted into a large engineering team building the control system for a Hot Strip Mill. **Fortran** was the language, **VAX/VMS** the Operating System. Shared memory, global variables, variable names implying the data types (Integer vs Real) were the norm. Considering the complexity of hot rolling of steel, starting from giant slabs to produce coils of steel at such tight tolerances of thickness and cooling rate - the tools were primitive demanding extreme skill and competence of the engineers.

The next generation called for stepping up the game and it was most opportune that the company decided to invest in **Ada** technology. VAX?VMS, VAX/ELN, Ada, DECnet were a perfectly matched toolset and the team was inducted into this world. The robustness of the language, toolset, testing and debugging tools available made lifelong converts of many of the engineers. Over the next decade or so, this application grew to well over half a million lines of code - arguably the earliest and the largest non-defense application of Ada.

As the system evolved incorporating **PC** architecture, **Unix** derivative realtime operating system LynxOS the core architecture stood its own and thus contributed to my strong and unshakeable belief in the benefits of **Ada**. As I migrated to medical device engineering, I benefited enormously from the lessons learned yet unable to implement the language directly. However **Ada** remained the tool of choice for personal enrichment as well as auxiliary support tools whichever team I worked on since then and continues to be so.

Beyond the availability of affordable compiler systems, a significant stumbling block in propagating the enthusiasm was the lack of code examples within reach of learners, playgrounds that enable experimentation regardless of the development environment. The versatility and applicability of **Ada** beyond defense applications was mostly theoretical in perception.

This work is an attempt to fill the gap. With annotated code examples, problems from diverse application domains are presented. The **projectlet**s are complete and provide enough scope for experimentation. Each chapter includes a paragraph of **Stretch** projects that can lead to full featured real applications. It is hoped that the examples and additional exercises whet the appetite of learners enough to invite them to incorporate **Ada** in their own applications and derive all the possible benefits.

1 Introduction

Welcome to **Ada Skills Sharpened**. You are a scientist, engineer or a computer science enthusiast who is looking to get an exposure to **Ada**. Perhaps with some prior programming experience in **C** or **C++** or maybe **Python**. Ada is a great choice - another tool to add to your arsenal. It may (or in most cases may not) be a language in your ***daytime job***. Does not matter. It is not the syntax or the IDE that is going to be beneficial to you. It is the discipline of engineering software with extreme precision, with proper separation of specifications and implementation, the strong data typing and the depth of help provided behind the scenes by the compiler and the runtime that you are going to take away.

The emphasis in this book will be in **doing**. Design and development **projectlet**s, to meet clear, simple objectives will be outlined with annotations on the programming language features that can be used to stitch a solution.

A **projectlet** is a small project that can be realized in a very short time. The functions can contribute to your personal library. Arguably with many limitations and perhaps not quite ready for worldwide application. But the techniques you learn will contribute to building robust tools given the requisite resources.

The learnings are not very specific to **Ada**. The specifications have been implemented in other programming languages as a learning method. **Go**, **C#**, **Python** are some of the other languages that have been used. In my professional life, I used to assign a small selection of these projectlets to new team members to sharpen their skills and to demonstrate proficiency before being assigned more challenging tasks.

I invite you to exercise the sample code provided, make suggested improvements and even tackle the stretch projects. If nothing else, it is a lot of fun. The sample solutions are not necessarily without issues; not the best way to get the results very likely. They can all use improvements. It would delight me no end to receive criticisms and even better an improved solution. Docker containers are provided as appropriate for even bolder experimentation.

1.1 Reference Implementation

All the annotated code fragments in this book can be accessed from one of the following repositories:

https://gitlab.com/ada23/toolkit.git
https://gitlab.com/RajaSrinivasan/TechAdaBook.git
https://gitlab.com/ada23/codemd.git
https://gitlab.com/ada23/bindings/adamosquitto.git
https://gitlab.com/ada23/fileres.git
https://rsrinivasan8@bitbucket.org/rsrinivasanada/docker.git

The book is accessible in multiple forms:

Comments, reviews, corrections may be submitted directly in the repositories or emailed to me at rsrinivasan@alumni.iitm.ac.in.

2 Hello

In the grand old tradition of learning programming, we start with our version of the **hello world**. Our journey commences with the understanding of the toolbox, particularly those features that scientific computation will rely upon. Printouts to the terminal, basic data types and their features, simple numerical functions are the subject of the projectlets in this chapter. Stretching a bit further, a review of the **array** as a fundamental data structure is studied.

2.1 Learning Objectives

- Understand the fundamental data types supported by the language.
- Text output of simple variables of diverse data types
- Characteristics of the data types like storage occupied, range of values
- Arrays and their handling

2.2 Data Types

2.2.1 Projectlet Specification

Printout the size in number of bits, range for fundamental data types.

2.2.2 Sample Runs

Integer Types

```
~/bin/dtypes int
```

```
dtypes dtypes.adb Compiled Dec 02 2024 05:14:54
--------- dtypes.Integer_Types
Integer                 Size            32 bits
Integer                 Range          -2147483648 ..  2147483647
Short_Integer           Size            16 bits
Short_Integer           Range          -32768 ..  32767
Short_Short_Integer     Size            8 bits
Short_Short_Integer     Range          -128 ..  127
Long_Integer            Size            64 bits
Long_Integer            Range          -9223372036854775808 ..  9223372036854775807
```

As shown above, integer datatypes of 8,16,32 and 64 bits are supported. For each of these types, some key parameters are also shown. For example with an 8 bit Integer, values are in the range -32768 .. 32767. In particular these integer types are signed.

Unsigned Integer Types

```
~//bin/dtypes uns
```

```
dtypes dtypes.adb Compiled Dec 02 2024 05:14:54
--------- dtypes.Unsigned_Types
Unsigned_32             Size            32 bits
Unsigned_32             Range           0 ..  4294967295
Unsigned_16             Size            16 bits
Unsigned_16             Range           0 ..  65535
Unsigned_8              Size            8 bits
Unsigned_8              Range           0 ..  255
Unsigned_64             Size            64 bits
Unsigned_64             Range           0 ..  18446744073709551615
Natural                 Size            31 bits
Natural                 Range           0 ..  2147483647
Positive                Size            31 bits
Positive                Range           1 ..  2147483647
```

Corresponding to each of the bit sizes, there are also unsigned integer types as shown above.

Real or Floating Types

```
~/bin/dtypes float
```

```
dtypes dtypes.adb Compiled Dec 02 2024 05:14:54
```

```
--------- dtypes.Float_Types
Float                   Size             32 bits
Float                   Range           -3.40282E+38 ..  3.40282E+38
Float                   Model_Small      1.17549E-38
Long_Float              Size             64 bits
Long_Float              Range           -1.79769313486232E+308 ..  1.79769313486232E+308
Float                   Model_Small      2.22507385850720E-308
Long_Long_Float         Size             64 bits
Long_Long_Float         Range           -1.79769313486232E+308 ..  1.79769313486232E+308
Float                   Model_Small      2.22507385850720E-308
```

Enumeration

In Ada, enumeration is a ***full fledged*** data type instead of being an alternate way of looking at groups of integers.

```
~/bin/dtypes enum
```

```
dtypes dtypes.adb Compiled Dec 02 2024 05:14:54
--------- dtypes.Enumeration_Types
Character               Size             8 bits
Character               Range           NUL .. 'ÿ'
Character <CR>          Internal         13
Character <CR>          Image           CR
Character $             Internal         36
Character $             Succ, Pred      % ; #
Boolean                 Size             1 bits
Boolean                 Machine_Size     8 bits
Day_Name                Size             3 bits
Day_Name                Machine_Size     8 bits
```

The above illustrates that **Character**, **Boolean** are enumerations as well as the names of Week Days. Of particular note is that the enumerations have a **Machine_Size** representing the number of bits allocated versus **Size** which is the minimum size required for this data type. These will become significant when data structures include members of these data types and the compiler has the option to optimize the storage.

String

In Ada the string is an array of Characters. Unlike **C** normal strings do not have a null termination requirement. Instead Strings are of fixed length determined at the time of creation of a string. This treatment of strings and other such data types is the cornerstone of ***safe programming practices*** imposed by the language.

```
~/bin/dtypes str
```

```
dtypes dtypes.adb Compiled Dec 02 2024 05:14:54
--------- dtypes.String_Type
Candidate String       : ~!@#$%^&*()_+`234567890-=
Length                 :  25
Size in bits           :  200
Truncate to first half : ~!@#$%^&*()_
Mid half               : ^&*()_+`2345
Multiply               : ~!@#$%^&*()_+`234567890-=~!@#$%^&*()_+`234567890-=
Occurrences of 'a'     :  0
Position of 'cal'      :  0
Head                   : ~!@#$%^&*()_+`234567890-=                       ;
Tail                   :                        ~!@#$%^&*()_+`234567890-=;
To Upper Case          : ~!@#$%^&*()_+`234567890-=
Candidate String       : alphabetical
Length                 :  12
Size in bits           :  96
Truncate to first half : alphab
Mid half               : habeti
Multiply               : alphabeticalalphabetical
Occurrences of 'a'     :  3
Position of 'cal'      :  10
Head                   : alphabetical                                    ;
Tail                   :                                     alphabetical;
To Upper Case          : ALPHABETICAL
Candidate String       : ~!@#$%^&*()_+`234567890-=alphabetical
Length                 :  37
Size in bits           :  296
Truncate to first half : ~!@#$%^&*()_+`2345
Mid half               : ()_+`234567890-=al
Multiply               : ~!@#$%^&*()_+`234567890-=alphabetical~!@#$%^&*()_+`234567890-=alpha
Occurrences of 'a'     :  3
Position of 'cal'      :  35
Head                   : ~!@#$%^&*()_+`234567890-=alphabetical           ;
Tail                   :            ~!@#$%^&*()_+`234567890-=alphabetical;
To Upper Case          : ~!@#$%^&*()_+`234567890-=ALPHABETICAL
```

2.2.3 Implementation

In languages like **C** the limits of the data types are defined as language defined macros. E.G INT_MAX and INT_MIN for maximum and minimum of the **int** data type. In Ada, the vehicle to access such parameters is the concept of **attribute**s. As shown for the Integer data types, all data types support appropriate attributes:

```
~/bin/codemd ..//Prj/dtypes/src/dtypes.adb -x IntTypes -l
```

```
        Annotation ("Integer", "Size");
        Put (Integer'(Integer'Size)'Image);
        Put_Line (" bits");
        Annotation ("Integer", "Range");
        Put (Integer'(Integer'First)'Image);
        Put (" .. ");
        Put (Integer'(Integer'Last)'Image);
        New_Line;

        Annotation ("Short_Integer", "Size");
        Put (Integer'(Short_Integer'Size)'Image);
        Put_Line (" bits");
        Annotation ("Short_Integer", "Range");
        Put (Integer (Short_Integer'First)'Image);
        Put (" .. ");
        Put (Integer (Short_Integer'Last)'Image);
        New_Line;

        Annotation ("Short_Short_Integer", "Size");
        Put (Integer'(Short_Short_Integer'Size)'Image);
        Put_Line (" bits");
        Annotation ("Short_Short_Integer", "Range");
        Put (Integer (Short_Short_Integer'First)'Image);
        Put (" .. ");
        Put (Integer (Short_Short_Integer'Last)'Image);
        New_Line;

        Annotation ("Long_Integer", "Size");
        Put (Long_Integer'(Long_Integer'Size)'Image);
        Put_Line (" bits");
        Annotation ("Long_Integer", "Range");
        Put (Long_Integer (Long_Integer'First)'Image);
        Put (" .. ");
```

```
      Put (Long_Integer (Long_Integer'Last)'Image);
      New_Line;

```

The enumeration data type examples shown above are supported by the language with richer set of ***attribute***s:

```
~/bin/codemd ../Prj/dtypes/src/dtypes.adb -x EnumTypes -l
```

```
      Annotation ("Character", "Range");
      Put (Character (Character'First)'Image);
      Put (" .. ");
      Put (Character'(Character'Last)'Image);
      New_Line;

      Annotation ("Character <CR>", "Internal");
      Put (Integer'(Character'Pos (ASCII.CR))'Image);
      New_Line;

      Annotation ("Character <CR>", "Image");
      Put (Character'Image (ASCII.CR));
      New_Line;

      Annotation ("Character $ ", "Internal");
      Put (Integer'(Character'Pos ('$'))'Image);
      New_Line;

      Annotation ("Character $", "Succ, Pred");
      Put (Character'Succ ('$'));
      Put (" ; ");
      Put (Character'Pred ('$'));
      New_Line;

```

The String data type illustrates leads to questions about the runtime support for various string operations.

```
~/bin/codemd ../Prj/dtypes/src/dtypes.adb -x StrType -l
```

```
         Annotation ("Candidate String ");
         Put_Line (str);
         Annotation ("Length ");
         Put_Line (Positive'(str'Length)'Image);
         Annotation ("Size in bits ");
         Put_Line (Positive'(str'Size)'Image);
         Annotation ("Truncate to first half ");
         Put_Line (str (1 .. str'Length / 2));
         Annotation ("Mid half ");
         Put_Line (str (1 + str'Length / 4 .. 3 * str'Length / 4));
         Annotation ("Multiply ");
         Put_Line (str & str);
         Annotation ("Occurrences of 'a'");
         Put_Line (Fixed.Count (str, "a")'Image);
         Annotation ("Position of 'cal'");
         Put_Line (Fixed.Index (str, "cal")'Image);
         Annotation ("Head ");
         Put (Fixed.Head (str, 48));
         Put_Line (";");
         Annotation ("Tail ");
         Put (Fixed.Tail (str, 48));
         Put_Line (";");
         Annotation ("To Upper Case ");
         Put_Line (Fixed.Translate (str, Maps.Constants.Upper_Case_Map));
```

In addition to attributes, all data types have rich support in the predefined library. A small set of libraries is required to support this project.

```
~/bin/codemd ../Prj/dtypes/src/dtypes.adb -x Library -l
```

```
with Interfaces;  use Interfaces;
with Ada.Text_IO; use Ada.Text_IO;

with Ada.Command_Line; use Ada.Command_Line;
with Ada.Strings.Fixed;
with Ada.Strings.Maps.Constants;

with Ada.Calendar.Formatting;

with GNAT.Source_Info; use GNAT.Source_Info;
```

2.3 Data Structures - Arrays

We encountered a basic array in **String**s; Strings are an array of **Character**s which itself is of the data type Enumeration. Further examples of arrays of Integers, Floats and Unsigned Integers with varying number of elements is illustrated in this projectlet.

Integer Arrays

```
~/bin/arr int
```

```
arr arr.adb Compiled Nov 12 2024 05:54:52
--------- arr.Int_Array
Characters that occurred in the quote
Character ' '          :  19
Character ','          :  1
Character '.'          :  1
Character 'E'          :  1
Character 'S'          :  1
Character 'a'          :  10
Character 'b'          :  2
Character 'c'          :  1
Character 'd'          :  5
Character 'e'          :  16
Character 'f'          :  1
Character 'g'          :  4
Character 'h'          :  5
Character 'i'          :  10
Character 'k'          :  2
Character 'l'          :  5
Character 'm'          :  2
Character 'n'          :  11
Character 'o'          :  4
Character 'p'          :  2
Character 'r'          :  9
Character 's'          :  7
Character 't'          :  10
Character 'u'          :  5
Character 'w'          :  2
Character 'y'          :  1
```

Delared and processed by the code fragment:

```
~/bin/codemd ../Prj/dtypes/src/arr.adb -x Constrained -l
```

```
charcounts : array (Character'First .. Character'Last) of Integer :=
  (others => 0);
```

and

```
~/bin/codemd ../Prj/dtypes/src/arr.adb -x ForEachChar -l
```

Some important things to note are the array index which in this case is of type **Character**. As a result, the iteration over the entire array is syntactically interesting though conceptually normal.

Real Arrays

```
~/bin/arr real
```

```
arr arr.adb Compiled Nov 12 2024 05:54:52
--------- arr.Real_Array
Real Array             Length           4
                       Size             128 bits
                       Range            1 ..  4
Raise to the Power 3   :
 1   1.00000E+00     1.00000E+00
 2   1.00000E+01     1.00000E+03
 3   1.00000E+02     1.00000E+06
 4   1.00000E+03     1.00000E+09
```

implemented as:

```
~/bin/codemd ../Prj/dtypes/src/arr.adb -x ArrTypeFloat -l
```

```
type Real_Array is array (Integer range <>) of Float;
args : Real_Array := (1 => 1.0, 2 => 10.0, 3 => 100.0, 4 => 1_000.0);
procedure Show_Powers (vals : Real_Array; p : Integer) is
begin
   for v in vals'Range loop
      Put (v'Image);
      Put (ASCII.HT);
      Put (vals (v)'Image);
```

```
            Put (ASCII.HT);
            Put (Float'(vals (v)**p)'Image);
            New_Line;
         end loop;
      end Show_Powers;
```

This example illustrates constraining of the array (ie specifying the limits of the array) implicitly by initialization. The array limits are static though the contents can be changed in this example.

Unsigned Integer Arrays

```
~/bin/arr uns
```

```
arr arr.adb Compiled Nov 12 2024 05:54:52
--------- arr.Unsigned_Array
Candidate              :  7
                       Digitised        7,
                       Reconstructed    7
Candidate              :  99
                       Digitised        9,  9,
                       Reconstructed    99
Candidate              :  199
                       Digitised        1,  9,  9,
                       Reconstructed    199
Candidate              :  244
                       Digitised        2,  4,  4,
                       Reconstructed    244
```

implemented as :

```
~/bin/codemd ../Prj/dtypes/src/arr.adb -x ArrTypeUns -l
```

```
      type Unsigned_Array is array (Integer range <>) of Unsigned_8;
      subtype DecimalDigits_Array is Unsigned_Array (1 .. 11);
      candidates : Unsigned_Array (1 .. 4) := (7, 99, 199, 244);

      function Digitize (val : Unsigned_8) return DecimalDigits_Array is
         result : DecimalDigits_Array := (others => 0);
         dp     : Integer             := 0;
         valnow : Unsigned_8          := val;
```

```
   begin
      while valnow > 0 loop
         result (result'Last - dp) := valnow rem 10;
         dp                        := dp + 1;
         valnow                    := valnow / 10;
      end loop;
      return result;
   end Digitize;

   function Value (val : DecimalDigits_Array) return Unsigned_8 is
      result : Unsigned_8 := 0;
   begin
      for d in val'Range loop
         result := result * 10 + val (d);
      end loop;
      return result;
   end Value;

   procedure Show (vals : Unsigned_Array) is
      significant : Boolean := False;
   begin
      for v in vals'Range loop
         if significant then
            Put (vals (v)'Image);
            Put (", ");
         elsif vals (v) > 0 then
            significant := True;
            Put (vals (v)'Image);
            Put (", ");
         end if;
      end loop;
      if not significant then
         Put ("0");
      end if;
   end Show;
```

In this case, a function is defined that returns an **array type**. The arrays are passed around like any variable of any other data type and the callee can query the limits (though not the data types) at runtime.

2.4 Sample Implementation

```
Repository: TechAdaBook
Directory: Prj/dtypes
```

3 File Input/Output

Simple file handling is the focus of this chapter. Reading existing files is at the core of many applications and is illustrated with several examples. All the output in this chapter is to the default **standard output** postponing creation of files to a later chapter.

Code organization is a key to building reliable applications. With this in mind, key ideas of this chapter are organized as a reusable toolkit. The applications built are then a rather thin layer that use the library. The library itself will be grown over the entire book.

3.1 Learning Objectives

- Command Line processing
- Reading text files
- Simple string search
- Binary File Reading
- Simple Transformations of files
- Code library to enable reuse

3.2 Projectlet - fileio

3.2.1 Sample Runs

This projectlet will leverage the command line to combine diverse functions into one executable.

> ~/bin/fileio

```
fileio
usage: fileio [lc|ls|find] file1 file2 ...
```

Line Count of a set of files

```
~/bin/fileio lc ../Prj/fileio/fileio.gpr ../Prj/fileio/src/fileio.adb
```

```
fileio
fileio.linecount
File ../Prj/fileio/fileio.gpr has  22 lines
File ../Prj/fileio/src/fileio.adb has  146 lines
```

List a small file.

```
~/bin/fileio list ../Prj/fileio/fileio.gpr
```

```
fileio
fileio.list
*****../Prj/fileio/fileio.gpr
 1      : with "config/fileio_config.gpr";
 2      : project Fileio is
 3      :
 4      :    for Source_Dirs use ("src/", "config/");
 5      :    for Object_Dir use "obj/" & Fileio_Config.Build_Profile;
 6      :    for Create_Missing_Dirs use "True";
 7      :    for Exec_Dir use "../bin";
 8      :    for Main use ("fileio.adb");
 9      :
 10     :    package Compiler is
 11     :       for Default_Switches ("Ada") use Fileio_Config.Ada_Compiler_Switches;
 12     :    end Compiler;
 13     :
 14     :    package Binder is
 15     :       for Switches ("Ada") use ("-Es"); --  Symbolic traceback
 16     :    end Binder;
 17     :
 18     :    package Install is
 19     :       for Artifacts (".") use ("share");
 20     :    end Install;
 21     :
 22     : end Fileio;
***** End of File ******
```

```
~/bin/fileio find package ../Prj/fileio/fileio.gpr
```

```
fileio
fileio.search
*****../Prj/fileio/fileio.gpr
 10     :    package Compiler is
 14     :    package Binder is
 18     :    package Install is
***** Found  3 instances
```

3.2.2 Code Fragments

Predefined Language Environment

```
~/bin/codemd ../Prj/fileio/src/fileio.adb -x Library -l
```

```
0002 | with Ada.Text_IO;      use Ada.Text_IO;
0003 | with Ada.Command_Line; use Ada.Command_Line;
0004 | with Ada.Strings.Fixed;
0005 |
0006 | with GNAT.Source_Info; use GNAT.Source_Info;
```

Command Line Processing

```
~/bin/codemd ../Prj/fileio/src/fileio.adb -x Command -l
```

```
0127 |    if Argument_Count >= 1 then
0128 |       declare
0129 |          operation : constant String := Argument (1);
0130 |       begin
0131 |          if operation = "lc" or operation = "linecount" then
0132 |             linecount;
0133 |          elsif operation = "ls" or operation = "list" then
0134 |             list;
0135 |          elsif operation = "find" or operation = "search" then
0136 |             search;
0137 |          else
0138 |             Put (operation);
0139 |             Put_Line (" is not a supported command");
```

```
         end if;
      end;
```

Line Oriented text file input

```
~/bin/codemd ../Prj/fileio/src/fileio.adb -x ReadLines -l
```

```
         while not End_Of_File (txtfile) loop
            Get_Line (txtfile, line, linelength);
            line_number := line_number + 1;
            argpos := Strings.Fixed.Index (line (1 .. linelength), arg, 1);
            if argpos > 0 then
               count := count + 1;
               Put (line_number'Image);
               Set_Col (8);
               Put (" : ");
               Put_Line (line (1 .. linelength));
            end if;
         end loop;
```

3.2.3 Sample Implementation

A sample implementation of the projectlets is available from:

```
https://gitlab.com/RajaSrinivasan/TechAdaBook.git
Directory: Prj/fileio
```

3.3 Projectlet - dump

This projectlet reads files as a binary stream without regard to any line breaks. The data is **dump**'ed in hexadecimal format.

3.3.1 Sample Runs

In the first example, the command line argument is dumped in hexadecimal form. The output is in 3 columns - the byte offset from the beginning of the string, a printout in ASCII followed by hexadecimal representation.

```
~/bin/dump "An arbitrary string to dump in hex"
```

```
     16#0# * An.arbitrary.str            * 416e20617262697472617279207374 72 *
    16#10# * ing.to.dump.in.h            * 696e6720746f2064756d7020696e2068 *
    16#20# * ex                          * 6578                             *
```

A variation is to print out the contents of files.

```
~/bin/dumpf ~/profile.sh
```

```
Dump of /Users/rajasrinivasan/profile.sh Size :  283
      16#0# * ...bin.bash..exp            * 23212f62696e2f626173680a23657870 *
     16#10# * ort.PATH..HOME.t            * 6f727420504154483d24484f4d452f74 *
     16#20# * ools.bin.gnat.na            * 6f6f6c732f62696e2f676e61745f6e61 *
     16#30# * tive.11.2.3.f008            * 746976655f31312e322e335f66303038 *
     16#40# * a8a7.bin..HOME.t            * 613861372f62696e3a24484f4d452f74 *
     16#50# * ools.bin.gprbuil            * 6f6f6c732f62696e2f6770726275696c *
     16#60# * d.22.0.1.b1220e2            * 645f32322e302e315f62313232306532 *
     16#70# * b.bin..PATH.expo            * 622f62696e3a24504154480a6578706f *
     16#80# * rt.PATH..PATH..H            * 727420504154483d24504154483a2448 *
     16#90# * OME.bin..HOME.to            * 4f4d452f62696e3a24484f4d452f746f *
     16#A0# * ols.bin.export.G            * 6f6c732f62696e0a6578706f72742047 *
     16#B0# * PR.PROJECT.PATH.            * 50525f50524f4a4543545f504154483d *
     16#C0# * .GPR.PROJECT.PAT            * 244750525f50524f4a4543545f504154 *
     16#D0# * H..HOME.Prj.talk            * 483a24484f4d452f50726a2f74616c6b *
     16#E0# * .logging...Flutt            * 2f6c6f6767696e670a2320466c757474 *
     16#F0# * er.dev.export.PA            * 6572206465760a6578706f7274205041 *
    16#100# * TH..PATH..HOME.f            * 54483d24504154483a24484f4d452f66 *
    16#110# * lutter.bin.                 * 6c75747465722f62696e0a           *
```

3.3.2 Code Fragments

Considering that hexadecimal printouts are an important diagnostic tool for other applications, this is an opportunity to create a support library; finally wrapping the support into a cli tool for illustrative purposes.

The toolkit defines a set of packages **hex** and **hex.dump**. Command line tool **dump** to convert a string to hexadecimal and **dumpf** to dump an entire file are created. The applications themselves are almost trivial:

```
~/bin/codemd ~/Prj/GitLab/toolkit/examples/dump/src/dumpf.adb -x DumpFile -l
```

```
with Ada.Command_Line; use Ada.Command_Line ;
with hex.dump ;
procedure Dumpf is
begin
   hex.dump.Dump( Argument(1) );
end Dumpf ;
```

The core feature of this projectlet is the conversion of binary data block to a hexadecimal form. Basic building block:

```
~/bin/codemd ~/Prj/GitLab/toolkit/adalib/src/hex.adb -x BinHex -l
```

```
   function Image (bin : Interfaces.Unsigned_8) return Hexstring is
      use Interfaces;
      img     : Hexstring;
      Lnibble : constant Interfaces.Unsigned_8 := bin and 16#0f#;
      Hnibble : constant Interfaces.Unsigned_8 :=
        Interfaces.Shift_Right (bin and 16#f0#, 4);
   begin
      img (1) := Nibble_Hex (Integer (Hnibble) + 1);
      img (2) := Nibble_Hex (Integer (Lnibble) + 1);
      return img;
   end Image;
```

And the reverse - a hexadecimal string to a binary.

```
~/bin/codemd ~/Prj/GitLab/toolkit/adalib/src/hex.adb -x StrBin -l
```

```
   function Value (Hex : Hexstring) return Interfaces.Unsigned_8 is
      use Interfaces;
      Vhigh, Vlow : Interfaces.Unsigned_8;
   begin
      Vhigh := Value (Hex (1));
      Vlow  := Value (Hex (2));
      return Vhigh * 16 + Vlow;
   end Value;
```

These fragments can be integrated into a conversion of a block:

```
~/bin/codemd ~/Prj/GitLab/toolkit/adalib/src/hex.adb -x BlockHex -l
```

```
   function Image (binptr : System.Address; Length : Integer) return String is
      -- use Interfaces;
      img   : String (1 .. 2 * Length);
      bytes : array (1 .. Length) of Interfaces.Unsigned_8;
      for bytes'Address use binptr;
      hexstr : Hexstring;
   begin
      for binptr in 1 .. Length loop
         hexstr := Image (bytes (binptr));
         img (2 * (binptr - 1) + 1 .. 2 * (binptr - 1) + 2) := hexstr;
      end loop;
      return img;
   end Image;
```

And the reverse:

```
~/bin/codemd ~/Prj/GitLab/toolkit/adalib/src/hex.adb -x StrBlock -l
```

```
   function Value (Hex : String) return System.Storage_Elements.Storage_Array
   is
      use System.Storage_Elements;
      result : System.Storage_Elements.Storage_Array (1 .. Hex'Length / 2);
   begin
      for rptr in result'Range loop
         result (rptr) :=
           Value (Hex (2 * Integer (rptr) - 1 .. 2 * Integer (rptr)));
      end loop;
      --Put_Line (Hex);
      return result;
   end Value;
```

A file is then read as a block of binary data and converted into hex strings:

```
~/bin/codemd ~/Prj/GitLab/toolkit/adalib/src/hex-dump.adb -x DumpFile -l
```

```
      Ada.Streams.Stream_IO.Open
        (file, Ada.Streams.Stream_IO.In_File, filename);
      stream := Ada.Streams.Stream_IO.Stream (file);
      declare
         buffer    :
           Ada.Streams.Stream_Element_Array
             (1 .. Ada.Streams.Stream_Element_Offset (filesize));
         bufferlen : Ada.Streams.Stream_Element_Offset;
      begin
         stream.Read (buffer, bufferlen);
         if bufferlen > 0 then
            Hex.dump.Dump
              (buffer'Address, Integer (bufferlen), bare => bare,
               show_offset => show_offset, Blocklen => Blocklen,
               Outfile                                  => Outfile);
         end if;
      end;
```

3.4 Stretch

1. **Resource Packer:** the goal is to enable executables to include resources such as graphics files or other files as resources instead of external files. A word puzzle application for example can then use the resource at runtime instead of having to be given a file to load. The output expected is a source file in a suitable language such as **Ada** that can be compiled as a part of the application. Reference Implementation

3.5 Sample Implementation

```
Repository: toolkit
Directory: examples/dump
```

4 Vectors

A consequence of the strong typing model implemented in **Ada** even **String**s are to be of fixed size - requiring a declaration before being used. While reading text files, a String buffer is provided though it is almost certain that all the lines are not of the same length. Such variable length dynamic datastructures are a necessity yet the safety of the data structures at run time.

In this projectlet, a simple container library package **Ada.Containers.Vectors** is exploited as a way to provide dynamically sized arrays of e.g. Integers.

4.1 Learning Objectives

- Safe manipulation of dynamically sized arrays (vectors)
- Iteration
- Fixed Point datatype

4.2 Library foundation

The predefined library offers the **generic** package Ada.Containers.Vectors which needs to be **instantiate**d with a specific data type in order to support the dynamic arrays of that datatype. In our projectlet, we use the facility to create dynamic arrays of say ***octal***, ***decimal*** or ***hexadecimal*** digits. In addition dynamic arrays for **Natural** numbers is also created. The support is utilized for a variety of recreational mathematical algorithms.

4.3 Custom Vectors

```
~/bin/codemd ../../toolkit/adalib/src/numbers.ads -x Define -l
```

```
   subtype OctalDigit_Type is Natural range 0 .. 7;
   subtype DecimalDigit_Type is Natural range 0 .. 9;
   subtype HexadecimalDigit_Type is Natural range 0 .. 15;

   package OctalVector_Pkg is new Ada.Containers.Vectors
     (Natural, OctalDigit_Type);
   subtype OctalDigits_Type is OctalVector_Pkg.Vector;

   package DecimalVector_Pkg is new Ada.Containers.Vectors
     (Natural, DecimalDigit_Type);
   subtype DecimalDigits_Type is DecimalVector_Pkg.Vector;

   package HexadecimalVector_Pkg is new Ada.Containers.Vectors
     (Natural, HexadecimalDigit_Type);
   subtype HexadecimalDigits_Type is HexadecimalVector_Pkg.Vector;
```

As shown above we define 3 custom data types - all numerics but with different range specifications. Each then requires an instantiation of the basic Containers.Vectors. The created packages have the support for dynamic arrays of the appropriate data type. The HexadecimalVectors_Pkg supports vectors of HexadecimalDigit_Type elements.

```
~/bin/codemd ../../toolkit/adalib/src/numbers.ads -x Basics -l
```

```
   function Convert (value : Natural) return OctalVector_Pkg.Vector;
   function Convert (value : Natural) return DecimalVector_Pkg.Vector;
   function Convert (value : Natural) return HexadecimalVector_Pkg.Vector;
   function Value (digs : OctalVector_Pkg.Vector) return Natural;
   function Value (digs : DecimalVector_Pkg.Vector) return Natural;
   function Value (digs : HexadecimalVector_Pkg.Vector) return Natural;
   function Image (digs : OctalVector_Pkg.Vector) return String;
   function Image (digs : DecimalVector_Pkg.Vector) return String;
   function Image (digs : HexadecimalVector_Pkg.Vector) return String;
```

We add our helper functions built around the basic vectors as shown above.

```
~/bin/codemd ../../toolkit/adalib/src/numbers.adb -x Convert -l
```

```
   function Convert (value : Natural) return DecimalVector_Pkg.Vector is
      result : DecimalVector_Pkg.Vector;
      valnow : Natural := value;
   begin
```

```
      if valnow < 10 then
         result.Append (valnow);
         return result;
      end if;
      loop
         result.Prepend (valnow mod 10);
         valnow := valnow / 10;
         if valnow < 1 then
            exit;
         end if;
      end loop;
      return result;
   end Convert;
```

In the above example, a number is split into its decimal digits the result being a dynamically sized array. The reverse operation is performed by retrieving individual elements:

```
~/bin/codemd ../../toolkit/adalib/src/numbers.adb -x Value -l
```

```
   function Value (digs : DecimalVector_Pkg.Vector) return Natural is
      use DecimalVector_Pkg;
      result : Natural := 0;
      curs   : Cursor  := digs.First;
   begin
      loop
         result := result * 10 + Element (curs);
         if curs = Last (digs) then
            exit;
         end if;
         curs := Next (curs);
      end loop;
      return result;
   end Value;
```

Alternatively, the library and thus the instantiated version, supports a cleaner and more elegant way to iterate over the elements:

```
~/bin/codemd ../../toolkit/adalib/src/numbers.adb -x Iterate -l
```

```
   procedure ShowNumber (cursor : NumbersVector_Pkg.Cursor) is
   begin
```

```
      Put (NumbersVector_Pkg.Element (cursor)'Image);
      Put (", ");
   end ShowNumber;

   procedure Show (num : NumbersVector_Pkg.Vector) is
   begin
      num.Iterate (ShowNumber'Access);
   end Show;
```

4.4 Fixed Point Data type

We have so far dealt with variations of **Integer** and **Float** data types; neither of these are convenient for a notion such as abundance. Somewhat unique to **Ada** is the fixed point data type which can take on **discrete** values but in an arbitrary range including less than 1 and is a better fit.

```
~/bin/codemd ../../toolkit/adalib/src/numbers.ads -x Abundance -l
```

```
   -- Ref: https://mathworld.wolfram.com/Abundancy.html
   type AbundancyRatio is delta 0.01 digits 6;
   function Abundance (num : Natural) return AbundancyRatio;
```

This is fundamendal to identifying ***Multiperfect*** numbers for example.

4.5 Applications

Even a casual visit to a site like Wolframalpha inspires one to start excursions. With a foundational support for computing the list of ***divisor***s and ***factor***s many applications become far simpler.

```
~/bin/codemd ../../toolkit/adalib/src/numbers.adb -x Factors -l
```

```

   function DivisorSum (num : Natural) return Natural is
      use NumbersVector_Pkg;
      divs   : constant Vector  := Divisors (num);
      result : Natural := 0;
      procedure Summer (cur : Cursor) is
      begin
         result := result + Element (cur);
      end Summer;
   begin
      divs.Iterate (Summer'Access);
      return result;
   end DivisorSum;

   function Abundance (num : Natural) return AbundancyRatio is
      result : AbundancyRatio;
      divsum : constant Natural := DivisorSum (num);
   begin
      result := AbundancyRatio (Float (divsum) / Float (num));
      return result;
   end Abundance;

   function Factors (num : Natural) return NumbersVector_Pkg.Vector is
      result  : NumbersVector_Pkg.Vector;
      curfac  : Natural := 2;
      curnum  : Natural := num;
      sqrtnum : Natural;
   begin
      sqrtnum := 1 + Natural (Sqrt (Float (num)));
      if curfac > sqrtnum then
         result.Append (num);
         return result;
      end if;
      loop
         if curnum mod curfac = 0 then
            result.Append (curfac);
            curnum := curnum / curfac;
            if curnum <= 1 then
               exit;
            end if;
         else
            curfac := curfac + 1;
         end if;
      end loop;
      return result;
   end Factors;
```

```
~/bin/codemd ../../toolkit/adalib/src/numbers.adb -x Prime -l
```

```
   function IsPrime (num : Natural) return Boolean is
      use Ada.Containers, NumbersVector_Pkg;
      facs : constant Vector := Factors (num);
   begin
      if facs.Length > 1 then
         return False;
      end if;
      return True;
   end IsPrime;
```

4.5.1 Example usage

Unary algorithms

```
~/bin/num U 5749 8128 272
```

```
================= Number  5749
Octal        13165 Octal back conversion Ok
Decimal      5749 Decimal back conversion Ok
Hexadecimal 1675 HexaDecimal back conversion Ok
Divisors             1,  5749,
DivisorSum           5750
Prime Factors        5749,
Prime No            Yes
Perfect No          No
MultiPerfect No     No
Kaprekar No         No
================= Number  8128
Octal        17700 Octal back conversion Ok
Decimal      8128 Decimal back conversion Ok
Hexadecimal 1fc0 HexaDecimal back conversion Ok
Divisors             1,  2,  4,  8,  16,  32,  64,  127,  254,  508,  1016,  2032,  4064,  81
DivisorSum           16256
Prime Factors        2,  2,  2,  2,  2,  2,  127,
Prime No            No
Perfect No          Yes
MultiPerfect No     Yes
Kaprekar No         No
================= Number  272
Octal        420 Octal back conversion Ok
Decimal      272 Decimal back conversion Ok
```

```
Hexadecimal 110 HexaDecimal back conversion Ok
Divisors             1,  2,  4,  8,  16,  17,  34,  68,  136,  272,
DivisorSum           558
Prime Factors        2,  2,  2,  2,  17,
Prime No            No
Perfect No          No
MultiPerfect No     No
Kaprekar No         No
```

and the Binary algorithms

```
~/bin/num B 5749 8128 8128 272 5749 272 30 140 135 819 2480 6200 2480 30
```

```
========================= 5749 &  8128
GCD                  1
Friendly            FALSE
========================= 8128 &  272
GCD                  16
Friendly            FALSE
========================= 5749 &  272
GCD                  1
Friendly            FALSE
========================= 30 &  140
GCD                  10
Friendly            TRUE
========================= 135 &  819
GCD                  9
Friendly            TRUE
========================= 2480 &  6200
GCD                  1240
Friendly            TRUE
========================= 2480 &  30
GCD                  10
Friendly            TRUE
```

4.6 Stretch

- Distinctness - given a vector, develop an algorithm to determine if there are any duplicate members.

- Basic statistical support including functions for **mean**, **standard deviation**, **kurtosis**, **skewness** for vectors and **correlation coefficient** for multiple vectors will be a great lightweight addition to the toolkit.
- https://en.wikipedia.org/wiki/ISBN describes multiple forms of ISBN and the algorithms for calculating the checksum. A function that validates an ISBN - including the check digit is a reasonable exercise in extending the **numbers** package outlined in this projectlet.

4.7 Sample Implementation

```
Repository: toolkit
Directory: examples/num
```

5 Numerical Libraries

Ada features a strong predefined standard library covering a variety of disciplines. In particular the numerical libraries are a solid foundation for scientific and engineering computations. Combined with a comprehensive set of data structures lead to reliable applications.

In this chapter numerical libraries are surveyed.

5.1 Learning Objectives

- Creating text files in comma (or **:**) separated format
- Elementary numerical function library
- Interfacing with a plotting tool through files

5.1.1 Related Tools

In this projectlet, in addition to the library itself, we expand our toolset to other tools to create graphical representations. Basic charting and plotting is a powerful beginning to understanding. While generating the plots directly from Ada is possible, the usage may be distracting at this stage. There are of course many possible tools for plotting, **R**, **Julia**, **Octave** being some popular Open Source packages. In this book, **gnuplot** is used as the preferred tool for plotting data sets. The output from all the application software is written to a CSV file which then is fed to gnuplot to produce these plots.

5.2 Sinusoid

Basic sinusoids are supported directly by the library.

Code Fragment

```
~/bin/codemd ../Prj/dtypes/src/curves.adb -x DegRad -l
```

```
   function Radians (d : Integer) return Float is
   begin
      return 2.0 * Ada.Numerics.Pi * Float (d) / 360.0;
   end Radians;
   function Degrees (r : Float) return Integer is
   begin
      return Integer (r * 360.0 / (2.0 * Ada.Numerics.Pi));
   end Degrees;
```

Functions as defined above leverage the basic constants such as **pi** and **e** defined in the libraries.

```
~/bin/codemd ../Prj/dtypes/src/curves.adb -x Trig -l
```

```
      CreateOutputFile (myname, outfile);
      for deg in 0 .. 360 loop
         Put (Float'(Radians (deg))'Image);
         Put (" ; ");
         Put (Float'(Elementary_Functions.Sin (Radians (deg)))'Image);
         Put (" ; ");
         Put (Float'(Elementary_Functions.Cos (Radians (deg)))'Image);
         Put (" ; ");
         Put (Float'(Elementary_Functions.Tan (Radians (deg)))'Image);
         New_Line;
      end loop;
      Close (outfile);
```

which produces a CSV file in several columns :

```
~/bin/curves trig
sed -n "10,20p" curves.Trigonometric.csv
```

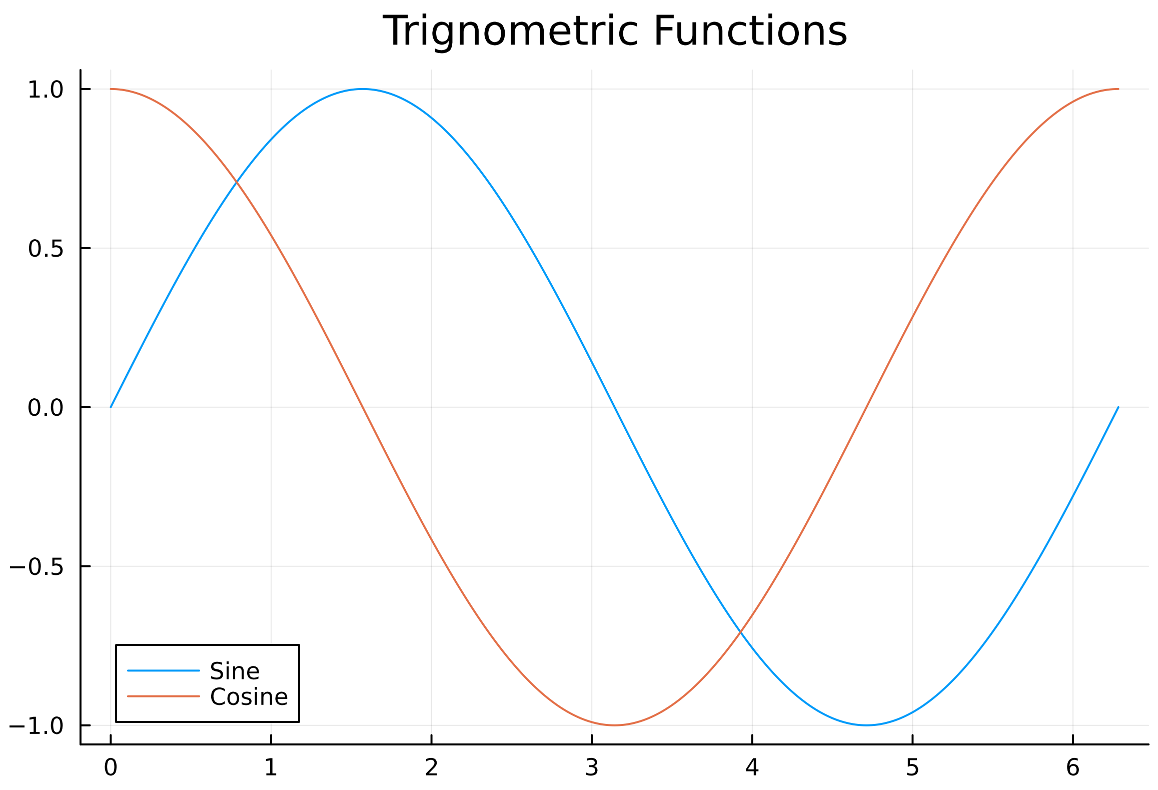
```
curves curves.adb Compiled Jan 21 2025 07:20:27
curves.Trigonometric
 1.57080E-01 ;  1.56434E-01 ;  9.87688E-01 ;  1.58384E-01
 1.74533E-01 ;  1.73648E-01 ;  9.84808E-01 ;  1.76327E-01
 1.91986E-01 ;  1.90809E-01 ;  9.81627E-01 ;  1.94380E-01
 2.09440E-01 ;  2.07912E-01 ;  9.78148E-01 ;  2.12557E-01
 2.26893E-01 ;  2.24951E-01 ;  9.74370E-01 ;  2.30868E-01
 2.44346E-01 ;  2.41922E-01 ;  9.70296E-01 ;  2.49328E-01
```

```
2.61799E-01 ;  2.58819E-01 ;  9.65926E-01 ;  2.67949E-01
2.79253E-01 ;  2.75637E-01 ;  9.61262E-01 ;  2.86745E-01
2.96706E-01 ;  2.92372E-01 ;  9.56305E-01 ;  3.05731E-01
3.14159E-01 ;  3.09017E-01 ;  9.51057E-01 ;  3.24920E-01
3.31613E-01 ;  3.25568E-01 ;  9.45519E-01 ;  3.44328E-01
```

which can be plotted using **Julia**

```
using CSV
using DataFrames
using Plots

df=DataFrames.DataFrame(CSV.File("curves.Trigonometric.csv",delim=";",header=false));
plot(df.Column1 , [df.Column2,df.Column3] ,
                    label=["Sine" "Cosine"] ,
                    title="Trignometric Functions" , show=true)
```



5.3 Hyperbolic

The numerical library supports hyperbolic functions as well:

```
~/bin/codemd ../Prj/dtypes/src/curves.adb -x Hyper -l
```

```
      CreateOutputFile (myname, outfile);
      for d in 0 .. 40 loop
         x := Float (d) * 0.1 - 2.0;
         Put (x'Image);
         Put (" ; ");
         Put (Elementary_Functions.Sinh (x), Fore => 3, Aft => 3, Exp => 0);
         Put (" ; ");
         Put (Elementary_Functions.Cosh (x), Fore => 3, Aft => 3, Exp => 0);
         Put (" ; ");
         Put (Elementary_Functions.Tanh (x), Fore => 3, Aft => 3, Exp => 0);
         Put (" ; ");
         New_Line;
      end loop;
      Close (outfile);
      Set_Output (Standard_Output);
```

In this fragment, it can be seen that the floating point variables are directly printed instead of using the built in **Image** attribute. Direct print then affords the control over the format of the print out. The output generated by the above fragment illustrates this:

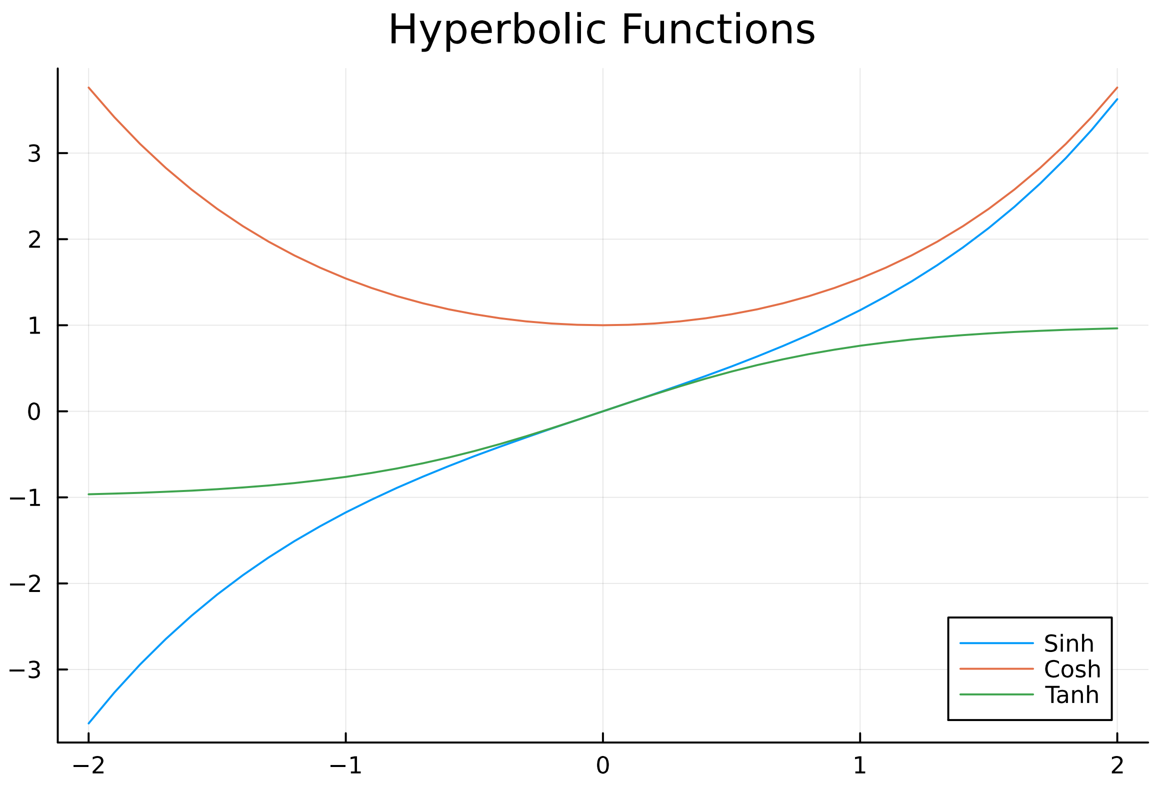
```
~/bin/curves hyper
sed -n "10,20p" curves.Hyperbolic.csv
```

```
curves curves.adb Compiled Jan 21 2025 07:20:27
curves.Hyperbolic
-1.10000E+00 ;  -1.336 ;   1.669 ;  -0.800 ;
-1.00000E+00 ;  -1.175 ;   1.543 ;  -0.762 ;
-9.00000E-01 ;  -1.027 ;   1.433 ;  -0.716 ;
-8.00000E-01 ;  -0.888 ;   1.337 ;  -0.664 ;
-7.00000E-01 ;  -0.759 ;   1.255 ;  -0.604 ;
-6.00000E-01 ;  -0.637 ;   1.185 ;  -0.537 ;
-5.00000E-01 ;  -0.521 ;   1.128 ;  -0.462 ;
-4.00000E-01 ;  -0.411 ;   1.081 ;  -0.380 ;
-3.00000E-01 ;  -0.305 ;   1.045 ;  -0.291 ;
```

```
-2.00000E-01 ;  -0.201 ;   1.020 ;  -0.197 ;
-1.00000E-01 ;  -0.100 ;   1.005 ;  -0.100 ;
```

```
using CSV
using DataFrames
using Plots

df=DataFrames.DataFrame(CSV.File("curves.Hyperbolic.csv",delim=";",header=false));
plot(df.Column1 , [df.Column2,df.Column3,df.Column4] ,
                    label=["Sinh" "Cosh" "Tanh"] ,
                    title="Hyperbolic Functions" , show=false)
```



5.4 Hypotrochoid

The curve **hypotrochoid** and its parameteric equations are found in Wikipedia

Code Fragment

```
~/bin/codemd ../Prj/dtypes/src/curves.adb -x Hypo -l
```

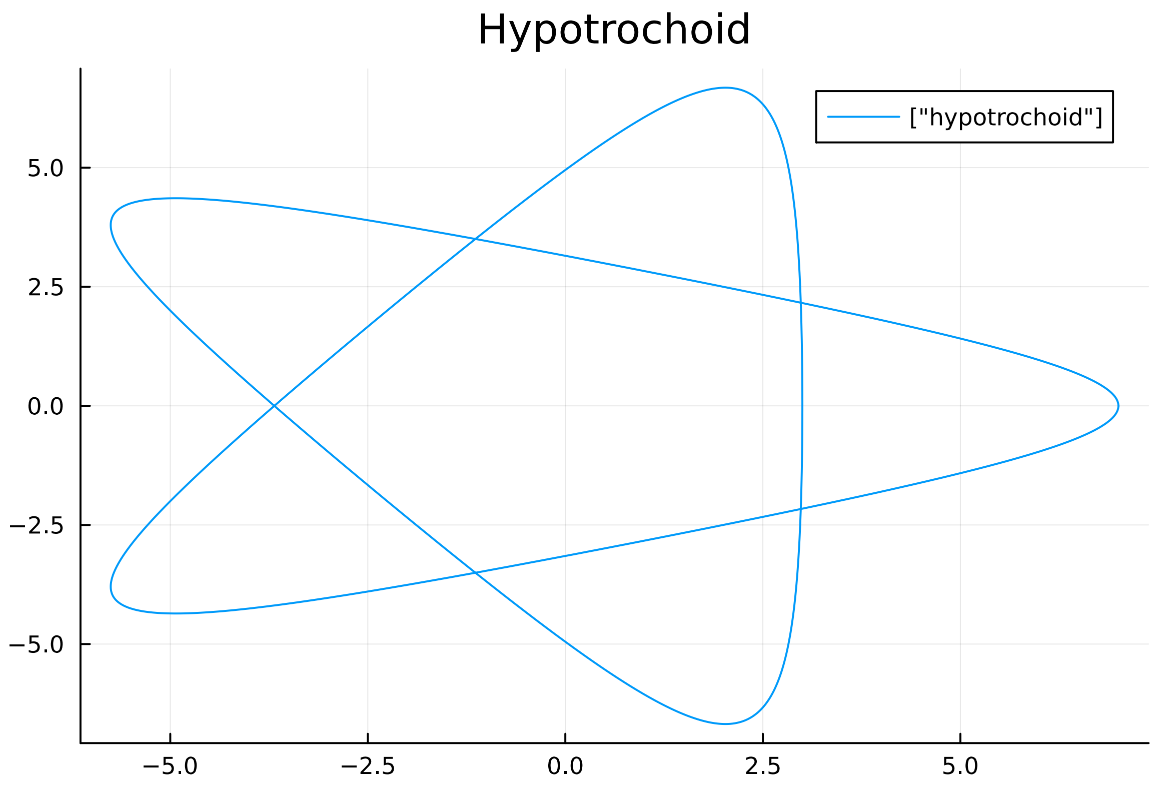
```
0123 |       for theta in 0 .. 3 * 360 loop
0124 |          x :=
0125 |            (fixedR - rollingR) * Cos (Radians (theta)) +
0126 |            d * Cos (Radians (theta) * (fixedR - rollingR) / rollingR);
0127 |          y :=
0128 |            (fixedR - rollingR) * Sin (Radians (theta)) -
0129 |            d * Sin (Radians (theta) * (fixedR - rollingR) / rollingR);
0130 |          Put (Float (Radians (theta))'Image);
0131 |          Put (" ; ");
0132 |          Put (x'Image);
0133 |          Put (" ; ");
0134 |          Put (y'Image);
0135 |          New_Line;
0136 |       end loop;
```

```
~/bin/curves troch
```

```
curves curves.adb Compiled Jan 21 2025 07:20:27
curves.Hypotrochoid
```

```
using CSV
using DataFrames
using Plots

df=DataFrames.DataFrame(CSV.File("curves.Hypotrochoid.csv",delim=";",header=false));
plot(df.Column2 , [df.Column3] ,
                  label=["hypotrochoid"] ,
                  title="Hypotrochoid" , show=false)
```



5.5 Stretch

1. Step response in the time domain gives an overview of the step response in the time domain of a capacitor and a resistor. Given a circuit with a resistor and a capacitor in series, generate the voltages across each component when a DC voltage is applied in a CSV file to enable plotting.

5.6 Sample Implementation

```
Repository: TechAdaBook
Directory: Prj/dtypes
```

6 Date and Time Handling

The topic for this chapter is Date and Time. Deceptively simple, handling dates and times in the context of leap years, across time zones and daylight savings and other such phenomena make this extremely complicated. In a projectlet or 2, we will explore some problems and thus exercise the predefined libraries of the language.

6.1 Date of Birth

6.1.1 Learning Objectives

- Convert Date and Time between internal and Printable formats
- Arithmetic with Date, Time and Duration

6.1.2 Projectlet Specification

Given a date of birth argument, compute and printout the age, details of the 50th, 60th and other significant birthdays. Leap years should be handled appropriately.

6.1.3 Sample Runs

```
~/bin/bday
```

```
Bday.P1
Date of Birth          1986-04-01 06:00:01
Day of the Week        TUESDAY
Age (days)              14191
   (years)              38
Birthday at Age  50    2036-04-01 04:00:00 TUESDAY
Birthday at Age  60    2046-04-01 05:00:00 SUNDAY
Birthday at Age  70    2056-04-01 05:00:00 SATURDAY
Birthday at Age  80    2066-04-01 04:00:00 THURSDAY
Birthday at Age  90    2076-04-01 04:00:00 WEDNESDAY
Birthday at Age  100   2086-04-01 04:00:00 MONDAY
```

6.1.4 Implementation

Predefined Libraries

```
~/bin/codemd ../Prj/datetime/bday/src/bday.adb -x Environment -l
```

```
with Ada.Calendar;             use Ada.Calendar;
with Ada.Calendar.Formatting;
with Ada.Calendar.Arithmetic; use Ada.Calendar.Arithmetic;
```

Date of Birth Computations

```
~/bin/codemd ../Prj/datetime/bday/src/bday.adb -x Calc -l
```

```
   procedure Show_Birthday (dob : Ada.Calendar.Time; age : Positive) is
      reqbday : Ada.Calendar.Time;
   begin
      reqbday := Time_Of (Year (dob) + age, Month (dob), Day (dob));

      Put ("Birthday at Age ");
      Put (Positive'Image (age));
      Set_Col (24);
      Put (Formatting.Image (reqbday));
      Set_Col (44);
      Put (Formatting.Day_Of_Week (reqbday)'Image);
      New_Line;
   end Show_Birthday;
```

6.2 Multiple Dates

6.2.1 Learning Objectives

- Date Comparison
- Passage of Time

6.2.2 Projectlet Specifications

- Given 2 dates, determine which is older. Essentially date comparisons.
- Assuming these are dates of birth, determine the year when the older age will be double the younger. Triple the younger.

6.2.3 Sample runs

```
~/bin/dates 1996-10-07
```

```
dates
dates.P1
Person A
    Birthday              1986-04-01 06:00:01
    Day of the Week       TUESDAY
    Age today              14191 days
                        38 years
Person B
    Birthday              1996-10-07 06:00:01
    Day of the Week       MONDAY
    Age today              10349 days
                        28 years
Person A is older
Age difference is  3842 days
               or  10 years
Double the Age
On 2006-10-07 04:00:00
Person B
 will be                10 years old
Person A
 will be                20 years old
```

6.2.4 Implementation

6.2.4.1 Code fragments

Given 2 dates of birth determine who is older. Then project when the older person will be double the age of the younger person.

```
~/bin/codemd ../Prj/datetime/bday/src/dates.adb -x Compare -l
```

```
procedure AgeDiff (operson : String; od : Time; yperson : String; yd : Time)
is
   agediff   : Day_Count;
   yearsdiff : Integer;
   dblage    : Time := od;
begin
   Put ("Person ");
   Put (operson);
   Put_Line (" is older");
   agediff   := yd - od;
   yearsdiff := Integer (agediff / days_in_year);
   Put ("Age difference is ");
   Put (agediff'Image);
   Put_Line (" days");
   Put ("               or ");
   Put (yearsdiff'Image);
   Put_Line (" years");
   Put_Line ("Double the Age");
   dblage := Age (yd, yearsdiff);

   Put ("On ");
   Put_Line (Formatting.Image (dblage));
   Show_Age (yperson, yd, dblage);
   Show_Age (operson, od, dblage);
end AgeDiff;
```

Add a number of years to a date of birth.

```
~/bin/codemd ../Prj/datetime/bday/src/dates.adb -x AddYears -l
```

```
function Age (dob : Time; years : Integer) return Time is
   agedTime : Time := dob;
begin
   agedTime :=
     Time_Of (Year_Number (Year (dob) + years), Month (dob), Day (dob));
   return agedTime;
end Age;
```

6.3 Timespan

6.3.1 Learning Objectives

- Sleep or delay execution for a specified length of time
- Set an alarm to wake up at a specific time
- Creating a widget to help identify performance sensitive sections of code

6.3.2 Sample Runs

Sleep for a specified length of time

```
~/bin/realtime p1
```

```
realtime
realtime.P1
Sleeping for  3.000000000 seconds
Woke up
Slept from 2025-02-06 10:24:45 to 2025-02-06 10:24:48
```

Sleep till a specified time or set an alarm

```
~/bin/realtime p2
```

```
realtime
realtime.P2
Sleeping till 2025-02-06 10:24:57 seconds
Woke up
Slept from 2025-02-06 10:24:48 to 2025-02-06 10:24:57
```

Perform computations and measure the time it takes for different count of cycles.

```
~/bin/codemd ../Prj/datetime/bday/src/realtime.adb -x Time -l
```

```
      Put ("Starting computations ");
      Put (iterations'Image);
      Put (" cycles");
      t1 := Clock;

      for i in 1 .. iterations loop
         result := Cosh (Float (i) * vf) + Sinh (Float (i) * vf);
      end loop;
      t2 := Clock;
      Put ("Completed ");
      Put (Duration'Image (t2 - t1));
      Put_Line (" seconds");
```

```
~/bin/realtime p3 10000
~/bin/realtime p3 20000
```

```
realtime
realtime.P3
Starting computations  10240000 cyclesCompleted  0.061613000 seconds
realtime
realtime.P3
Starting computations  20480000 cyclesCompleted  0.108192000 seconds
```

6.4 Sample Implementation

```
Repository: TechAdaBook
Directory: Prj/datetime
```

7 Strong Types

It is time to get serious about strong data types - one of the key strengths of Ada. We got a flavor of this with Strings of fixed length. In this chapter new data types which are application appropriate will be explored. In particular mechanisms to protect variables of strong data types will be studied.

7.1 Learning Objectives

- Strong Real data types.
- Checking values against type specifications
- Fixed Point data types
- Exceptions and their handling
- Record types

7.2 Projectlet - foods

Taking advantage of ***Strong Typing*** starts with application meaningful data types.

```
~/bin/codemd ../Prj/hello/src/foods.adb -x StrongType -l
```

```
    -- Ref: https://en.wikipedia.org/wiki/Glycemic_index
    type Glycemic_Index is range 0 .. 100;
    subtype Low_GI is Glycemic_Index range 0 .. 55;
    subtype Medium_GI is Glycemic_Index range 56 .. 69;
    subtype High_GI is Glycemic_Index range 70 .. 100;

    -- https://www.verywellhealth.com/glycemic-index-chart-for-common-foods-1087476
    Apple_GI      : constant Glycemic_Index := 36;
    Banana_GI     : constant Glycemic_Index := 51;
    Chocolate_GI  : constant Glycemic_Index := 40;
```

```
   SoyaBeans_GI  : constant Glycemic_Index := 16;
   Muesli_GI     : constant Glycemic_Index := 57;
   WheatRoti_GI  : constant Glycemic_Index := 62;
   Watermelon_GI : constant Glycemic_Index := 76;
   Cornflakes_GI : constant Glycemic_Index := 81;
```

A base type **Glycemic_Index** is defined. Different ranges within this base type are sub classes. Also defined are several constants in this basic type. Similarly:

```
~/bin/codemd ../Prj/hello/src/foods.adb -x StrongType2 -l
~/bin/codemd ../Prj/hello/src/foods.adb -x Vars -l
```

```
   type Glycemic_Load is range 0 .. 100;
   subtype Low_GL is Glycemic_Load range 0 .. 10;
   subtype Medium_GL is Glycemic_Load range 11 .. 19;
   subtype High_GL is Glycemic_Load range 20 .. 100;
   breakfast : Glycemic_Load;
   lunch     : Glycemic_Load;
   snack     : Glycemic_Load;
   dinner    : Glycemic_Load;
```

And computations and assignments involving these variables:

```
~/bin/codemd ../Prj/hello/src/foods.adb -x Compute -l
```

```
   breakfast := Orange_GL + WhiteBread_GL;
   lunch     := breakfast + breakfast;
   snack     := Load (Watermelon_GI, 10.0);
   dinner    := Load (SoyaBeans_GI, 15.0);
```

However, assignments cannot be arbitrary. For example, in the following, the compiler has enough information to prevent invalid assignments and will not compile the code. No quiet promotions or conversions of constants/variable types is probably difficult in the beginning but produces reliable softtware at the end.

```
~/bin/codemd ../Prj/hello/src/foods.adb -x Error1 -l
~/bin/codemd ../Prj/hello/src/foods.adb -x Error2 -l
```

```
0103 |
0104 |    -- error: invalid operand types for operator "+"
0105 |    -- error: left operand has type "Glycemic_Load" defined at line 30
0106 |    -- error: right operand has type "Glycemic_Index" defined at line 15
0107 |    -- lunch := Spaghetti_GL + Soyabeans_GI ;
0108 |
0112 |
0113 |    -- error: invalid operand types for operator "*"
0114 |    -- error: left operand has type "Glycemic_Load" defined at line 30
0115 |    -- error: right operand has type universal real
0116 |    -- lunch := breakfast * 2.0 ;
0117 |
```

In addition, it is also very straightforward to ascertain the variable value ranges at runtime:

```
~/bin/codemd ../Prj/hello/src/foods.adb -x Classify -l
```

```
0069 |    procedure Classify (glname : String; gl : Glycemic_Load) is
0070 |    begin
0071 |       Annotation (glname);
0072 |       if gl in Low_GL'Range then
0073 |          Put_Line ("Low_GL");
0074 |       elsif gl in Medium_GL'Range then
0075 |          Put_Line ("Medium_GL");
0076 |       elsif gl in High_GL'Range then
0077 |          Put_Line ("High_GL");
0078 |       else
0079 |          Put_Line ("Unable to determine");
0080 |       end if;
0081 |    end Classify;
```

At runtime

Having defined rules of data types, can we query their attributes at runtime? How are these data types implemented?

```
~/bin/foods
```

```
foods
Classification of Variable Values
Orange                 : Low_GL
```

```
WhiteBread              : Low_GL
Spaghetti               : High_GL

Variable Values
breakfast               :  15
lunch                   :  30
snack                   :  8
dinner                  :  2

Implementation Details
Glycemic_Index          Size             7 bits
                        Machine_Size     8 bits
                        Range            0 ..  100
```

7.3 Projectlet - blood_chem

Lest we conclude that new types and subtypes have to be built on integer types :

```
~/bin/codemd ../Prj/hello/src/blood_chem.adb -x Dtype -l
```

```
type Glucose_Concentration_Type is new Float range 0.0 .. 1_000.0;
subtype Hyperglycemic_Glucose_Type is
  Glucose_Concentration_Type range 140.0 .. 1_000.0;
subtype Hypoglycemic_Glucose_Type is
  Glucose_Concentration_Type range 0.0 .. 70.0;
subtype Euglycemic_Glucose_Type is
  Glucose_Concentration_Type range 70.0 .. 140.0;

my_glucose         : Glucose_Concentration_Type := 139.999_991;
good_glucose_value : Euglycemic_Glucose_Type;
hypo_value         : Hypoglycemic_Glucose_Type;
Hyper_value        : Hyperglycemic_Glucose_Type;
```

It is based on the basic *float* data type but *Glucose_Concentration_Type* is a new and distinct data type implying all the variable protections. In addition, with ranges specified for **subtype**s which are essentially different classes, the protection is a runtime expectation.

```
~/bin/codemd ../Prj/hello/src/blood_chem.adb -x Exception -l
```

```
0057 |    good_glucose_value := my_glucose;
0058 |    begin
0059 |       hypo_value := good_glucose_value;
0060 |    exception
0061 |       when Event : others =>
0062 |          Put_Line ("Assigning good_glucose_value to hypo_value");
0063 |          Put_Line (Ada.Exceptions.Exception_Message (Event));
0064 |    end;
```

Runtime

```
~/bin/blood_chem
```

```
Hypoglycemia Range is glucose levels below  7.00000E+01
Hyperglycemia Range is glucose levels above  1.40000E+02
My Glucose Level is  1.40000E+02
My Glucose levels are good
Assigning good_glucose_value to hypo_value
blood_chem.adb:62 range check failed
Assigning hypo_value to hyper_value
blood_chem.adb:71 range check failed
```

7.4 Sample Implementation

```
Repository: TechAdaBook
Directory: Prj/hello
```

8 Packages

It is time to get serious about code organization separating **Specifications** from **Implementation**. Though we did not dwell upon it, we encountered a package **hex** in the **dump** projectlet. In addition, the predefined language environment is made available to us primarily as packages.

In this chapter, a package to support **CSV** or Comma Separated Variable files is developed. The package is then applied to load and utilize a **foods** database leveraging the work done in an earlier chapter.

8.1 Learning Objectives

- Code Organization by packages
- Information hiding
- Richer Data Structures
- Container support in the Predefined Library

8.2 Projectlet - csv package

The separation of the **specification** of a feature from the details of implementation is central to the notion of packages's. By stating the interface specifications, the user or application is restricted in the interaction with the ***facility***. This affords the freedom to change the implementation e.g. more optimally or handling edge cases without imposing any changes on the user. In the following, a facility to support CSV files is specified:

```
~/bin/codemd ../../toolkit/adalib/src/csv.ads -x Spec -l
```

```
package Csv is
   Debug : Boolean := True;
   Duplicate_Column : exception;
   type File_Object_Type is limited private;
   type File_Type is access File_Object_Type;

   subtype Field_Count_Type is Integer range 1 .. 255;
   package String_Vectors_Pkg is new Ada.Containers.Indefinite_Vectors
     (Index_Type => Field_Count_Type, Element_Type => String);

```

```
...
```

```
private
   type File_Object_Type is limited record
      Session      : GNAT.AWK.Session_Type;
      No_Columns   : Integer := 0;
      Current_Line : Integer := -1;
      Field_Names  : String_Vectors_Pkg.Vector;
   end record;
end Csv;
```

In the above, the variable Debug is **public** and is thus can be read or even written by the user. Presumably the implementation uses this variable in some way but not specified.

An exception, a data type and a pointer to this data type are also declared. The application can then access these data types. However note that the ***File_Object_Type*** is designated private implying the details ie elements of this record are not accessible by the user.

The type **File_Type** is akin to a pointer though in Ada unlike **C**, this pointer can only point an object of tye type **File_Object_Type**. By default variables of this type will contain the value **NULL** which does not point to any object.

We also create a package String_Vectors_Pkg based on the template package Indefinite_Vectors one of quite a few **container** data structures supported by the language. The key data type from this package **Vector** is used to store the column names of the file.

Encountering for the first time a record data structure is similar to **C**. Of the members the **Session** draws from a library package **GNAT.awk** which is a programmatic interface supporting the **awk** semantics.

The interface specification or the contract provides the means to interact with the library:

```
~/bin/codemd ../../toolkit/adalib/src/csv.ads -x Interface -l
```

```
   function Open
     (Name : String; Separator : String; FieldNames : Boolean := True)
      return File_Type;
   procedure Close (File : in File_Type);
   function No_Columns (File : File_Type) return Integer;
   function Field_Name (File : File_Type; Column : Integer) return String;
   procedure Get_Line (File : in File_Type);
   function Line_No (File : File_Type) return Integer;
   function End_Of_File (File : File_Type) return Boolean;
   function Field (File : File_Type; Column : Integer) return String;
   function Field (File : File_Type; Name : String) return String;
   procedure Set_Names (file : File_Type; names : String_Vectors_Pkg.Vector);
   function Names (file : File_Type) return String_Vectors_Pkg.Vector;
```

The segments above should be sufficient to write an application using the facility as shown in the following fragment.

```
~/bin/codemd ../../toolkit/examples/meals/src/foods.adb -x CSVLoad -l
```

```
   function Load (mfn : String) return Meal_Type is
      use Ada.Strings.Unbounded;
      result   : Meal_TYpe;
      mealfile : csv.File_Type;
      dish     : Dish_Type;
   begin
      mealfile := csv.Open (mfn, ";");
      while not csv.End_Of_File (mealfile) loop
         csv.Get_Line (mealfile);
         dish.Name     := To_Unbounded_String (csv.Field (mealfile, 1));
         dish.Servings := Servings_Type'Value (csv.Field (mealfile, 2));
         result.Append (dish);
      end loop;
      csv.close (mealfile);
      return result;
   end Load;
```

The CSV file is **open**ed, **close**d and lines **Get_Line**d very similar to any text file. Additional functions retrieve the fields from each line. Field values are retrieved by their column number and returned as Strings. The user of CSV performs appropriate data conversions. The **meals** projectlet below puts the rest of the operation in context.

8.3 Projectlet: meals

In this projectlet, a food database is maintained as a CSV file. The database contains name, Glycemic Index, Glycemic Load, Calories and Protein content of food items. Once we have this, we can compute the total Calories of a meal containing the list of items along with the number of servings included. This could potentially be useful to determine Insulin bolus dosage for example for diabetics.

8.3.1 Requirements

- Primary Input is a CSV file detailing a meal. Each line indicates a food item and the number of servings of the item.
- Goal is to compute the total number of calories. This is done by referring to a foods database.
- Foods Database is a CSV file providing a list of food items, including their name, Serving Size, Calories per serving and other relevant information.

8.3.2 Example Usage

```
~/bin/meals ../../toolkit/examples/meals/breakfast.csv ../../toolkit/examples/meals/foods.
```

```
meals
Number of Fields  5
Field Name
Field Servings
Field Calories
Field Carbs
Field
Number of Fields  2
Field Name
Field  Servings
The meal is
Item Eggs   2 servings
Item OJ   1 servings
Item Coffee   1 servings
Item Toast   2 servings
Total Calories is  970
```

8.3.3 Detailed Design

The meal contents are structured as:

```
~/bin/codemd ../../toolkit/examples/meals/src/foods.ads -x Dish -l
```

```
   type Servings_Type is range 0 .. 8;
   type Dish_Type is record
      Name     : Ada.Strings.Unbounded.Unbounded_String;
      Servings : Servings_Type;
   end record;
```

A meal then is a collection of these.

Similarly the Foods Database items are structured as:

```
~/bin/codemd ../../toolkit/examples/meals/src/foods.ads -x FoodItem -l
```

```
   type Food_Item_Type is record
      Name        : Ada.Strings.Unbounded.Unbounded_String;
      ServingSize : Ada.Strings.Unbounded.Unbounded_String :=
        Ada.Strings.Unbounded.Null_Unbounded_String;
      Calories    : CaloriesType := CaloriesType'First;
      Proteins    : ProteinsType := ProteinsType'First;
      gi          : Glycemic_Index;
      gl          : Glycemic_Load;
      carbs       : CarbohydratesPerServing;
   end record;
   function Equal( Left, Right : Food_Item_Type ) return boolean ;
```

The foods database is a collection of these food items.

Collections

For both the foods database and each meal, a simple data structure like an array will be a good starting point. However, with the strong typing constraints, it becomes a bit challenging to size the arrays appropriately. A dynamically growing datastructure might be more suitable. **Ada** incorporate a rich library of **Container**s to choose from. The needs of this projectlet can be met by **Ada.Containers.Vector**s.

The container library provides templates of data structures or in **Ada** terminology **package**s from which application specific facilities are created by **instantiation** - a technical term for marrying the library specification with a specific data structure like so:

```
~/bin/codemd ../../toolkit/examples/meals/src/foods.ads -x Database -l
```

```
package FoodsDatabase_Pkg is new Ada.Containers.Vectors
  (Index_Type => Natural, Element_Type => Food_Item_Type , "=" => Equal);
subtype FoodsDatabase_Type is FoodsDatabase_Pkg.Vector;
function Load (filename : String) return FoodsDatabase_Type;
procedure Show (fi : Food_Item_Type);
procedure Show (db : FoodsDatabase_Type);
function Find(db : FoodsDatabase_Type ; item : String ) return Food_Item_Type ;
```

In this instance, a custom **package** has been designed - **FoodsDatabase__Pkg** which in turn created a datatype **FoodsDatabase__Pkg.Vector** which serves as the database of foods. Further API can be specified around these specifications as illustrated. The key function is the **Find** which returns the full details from the database based on the item name.

Using the same generic package, a distinct package is created for the meal itself like so:

```
~/bin/codemd ../../toolkit/examples/meals/src/foods.ads -x Meals -l
```

```
package Dishes_Pkg is new Ada.Containers.Vectors
  (Index_Type => Natural, Element_Type => Dish_Type);
subtype Meal_Type is Dishes_Pkg.Vector;
function Load (mfn : String) return Meal_Type;
procedure Add
  (meal : in out Meal_Type; item : String; servings : Servings_Type);
procedure Show (meal : Meal_Type);

function Calories (db : FoodsDatabase_Type ; meal : Meal_Type) return CaloriesType
```

The main requirement of the program namely the computation of the total caloric content of a meal is to be found in the **body** of the package foods. This body then is the home for the implementation of all the **procedure**s and **function**s comprising the package.

```
~/bin/codemd ../../toolkit/examples/meals/src/foods.adb -x Calories -l
```

```
function Calories (db : FoodsDatabase_Type ; dish : Dish_TYpe) return CaloriesType
 use FoodsDatabase_Pkg;
 fi : Food_Item_Type ;
 found : FoodsDatabase_Pkg.Cursor ;
begin
```

```
 fi.Name := dish.Name ;
 found := FoodsDatabase_Pkg.Find(db,fi);
 if found = FoodsDatabase_Pkg.No_Element
 then
    raise DatabaseError ;
 end if;
 return FoodsDatabase_Pkg.Element(found).Calories ;
end Calories ;

 function Calories (db : FoodsDatabase_Type ; meal : Meal_TYpe) return CaloriesType
    use Dishes_Pkg ;
    result : CaloriesType := 0 ;
    ptr  : Dishes_Pkg.Cursor;
 begin
    ptr := meal.First;
    while ptr /= Dishes_Pkg.No_Element loop
       result := result + CaloriesType( Float(Dishes_pkg.Element(ptr).Servings) *
                                        Float(Calories(db,Dishes_Pkg.Element(ptr)))
       ptr := Dishes_Pkg.Next(ptr);
    end loop ;
    return result ;
 end Calories;
```

8.4 Refinement

The sample implementation provided is sufficient for pedagogic purposes but is not robust enough which is left as an exercise. Some potential improvements that will belong in a **real** application are:

- Case insensitivity for food item names
- Robustness in handling malformed, corrupted database files
- Foods Database file location. Currently the default specification does not include any directory specification for the database file or the meals file itself

8.5 Stretch

1. Design a Project activities notebook. Over the day, one might be working on several projects and related activities. A simple command to keep track of activies of the day (or week) and be able to summarize them can be built patterned after this project.

2. Enhance **CSV** to support generation and create a table of computed formulae that can be fed into **gnuplot** for plotting. For example individual meals of the entire week may be summarized and the source of calories can be plotted as a **histogram**.

8.6 Sample Implementation

```
Repository: toolkit
Example: examples/foods
```

9 Text Processing

In this chapter we will tackle simple text processing.

9.1 Learning Objectives

- Command Line processing
- Regular Expressions and Pattern Matching
- Exceptions and error handling
- Pointers
- File system

9.2 Projectlet - vrex

There are quite a few resources e.g. https://regex101.com that help understand regular expressions better. For anyone tasked with validating say a phone number or postal code with international applications, such tools help experiment. This project builds such a tool but without any of the scaffolding like the web etc. A command line utility that can validate a string against a regular expression. As an extension, validate all the lines in a file against this pattern.

```
~/bin/vrex -h
```

```
vrex 0.1 Nov 25 2024 05:48:52
Usage: vrex regex [<string>] or -f  file1 file2 ,,.

 -v, --verbosity ARG    Verbosity Level
 -f, --files            Validate lines in the input file(s)
 -g, --glob             Glob Option
 -c, --case-insensitive Case insensitive. (Default sensitive)
```

Simple usage examples:

```
~/bin/vrex procedure Procedure
```

```
Does not match
```

The above does not **match** since case sensitivity is the default option. When we ignore the case, the match is successful as shown:

```
~/bin/vrex -c procedure Procedure
```

```
Matches
```

On the other hand we could accept either case for just the first character:

```
~/bin/vrex "[pP]rocedure" Procedure
```

```
Matches
```

And match each line of a file:

```
~/bin/vrex "[pP]rocedure" -f ../../toolkit/examples/vrex/src/*.ad*
```

```
Pattern [pP]rocedure ../../toolkit/examples/vrex/src/cli.adb
 0 lines matched
Pattern [pP]rocedure ../../toolkit/examples/vrex/src/cli.ads
 0 lines matched
Pattern [pP]rocedure ../../toolkit/examples/vrex/src/impl.adb
 0 lines matched
Pattern [pP]rocedure ../../toolkit/examples/vrex/src/impl.ads
 0 lines matched
Pattern [pP]rocedure ../../toolkit/examples/vrex/src/vrex.adb
 0 lines matched
```

In comparison, since **vrex** matches entire lines:

```
~/bin/vrex ".*[pP]rocedure.*" -f ../../toolkit/examples/vrex/src/*.ad*
```

```
Pattern .*[pP]rocedure.* ../../toolkit/examples/vrex/src/cli.adb
 16    :    procedure StringArg (name : String; ptr : GNAT.Strings.String_Access) is
 27    :    procedure Show_Arguments is
 33    :    procedure SwitchHandler
 47    :    procedure ProcessCommandLine is
 4 lines matched
Pattern .*[pP]rocedure.* ../../toolkit/examples/vrex/src/cli.ads
 15    :    procedure ProcessCommandLine;
 1 lines matched
Pattern .*[pP]rocedure.* ../../toolkit/examples/vrex/src/impl.adb
 20    :     procedure Matches( pattern : String ; filename : String ;
 1 lines matched
Pattern .*[pP]rocedure.* ../../toolkit/examples/vrex/src/impl.ads
 5     :     procedure Matches( pattern : String ; filename : String ;
 1 lines matched
Pattern .*[pP]rocedure.* ../../toolkit/examples/vrex/src/vrex.adb
 4     : procedure Vrex is
 1 lines matched
```

9.2.1 Implementation

The requirements are simple enough and the predefined library supports the package **GNAT.RegExp** which has the necessary support. So the environment is rather sparse:

```
~/bin/codemd ../../toolkit/examples/vrex/src/impl.adb -x Environment -l
```

```
0002 | with Ada.Text_Io; use Ada.Text_Io ;
0003 | with GNAT.RegExp ;
```

To match a string argument:

```
~/bin/codemd ../../toolkit/examples/vrex/src/impl.adb -x RegExp -l
```

```
0010 |     function Matches( pattern : String ; line : String ;
0011 |                       glob : boolean := false ; caseinsensitive : boolean := false ) i
0012 |         exp : GNAT.RegExp.RegExp := GNAT.RegExp.Compile( pattern , glob , not caseins
0013 |     begin
0014 |         return GNAT.RegExp.Match( line , exp  );
0015 |     end Matches ;
```

And match each line of a file:

```
~/bin/codemd ../../toolkit/examples/vrex/src/impl.adb -x ExpCompile -l
```

```
    pcompiled : access GNAT.RegExp.RegExp ;
    procedure Matches( pattern : String ; filename : String ;
                       glob : boolean := false ;
                       caseinsensitive : boolean := false) is
        file : File_Type ;
        line : String(1..MAX_LENGTH);
        linelength : Natural ;
        lineno : Natural := 0 ;
        count : Natural := 0 ;
    begin
        if pcompiled = null
        then
            pcompiled := new Gnat.RegExp.RegExp ;
            pcompiled.all := Gnat.RegExp.Compile( pattern , glob , not caseinsensitiv
        end if ;

...

            Get_Line(file,line,linelength);
            lineno := lineno + 1 ;
            if GNAT.RegExp.Match( line(1..linelength) , pcompiled.all  )
            then
                count := count + 1 ;
                Put(lineno'Image);
                Set_Col(8);
                Put(": ");
                Put_Line(line(1..linelength));
            end if ;

...

        Put(count'Image);
        Put_Line(" lines matched");
    end Matches ;
```

Note the compilation of the RegExp just once - the first time. The variable **pcompiled** is an **access** type variable - which is a pointer but to an object of a specific data type **GNAT.RegExp** in this case. Assignments to it can only be pointers to other GNAT.RegExp objects. The default value of this and all access variables is the special value **NULL**. Internally NULL may be any value but does not point to any object.

An assumption is made in this projectlet that the search is for the same expression in a series of files.

9.3 Projectlet - search

The goal of this projectlet is a utility to search text files for either a candidate string or a **Regular Expression**; optionally the candidate can be replaced by a replacement string. This can for example be used to convert all occurrences of e.g. PROcedure to procedure. The output of such a replacement is created as a new file (not overriding the input) in a specified directory. The entire operation is driven by the command line:

```
~/bin/frep -h
```

```
frep 0.1 Nov 24 2024 04:31:39
Usage: frep -s <string> -r <regexp> -R <replacement> -o outputdir <string> file1 file2 ,,.

 -v, --verbosity ARG  Verbosity Level
 -s, --search ARG     Search Candidate
 -r, --regex ARG      Search Candidate - Reg Exp
 -R, --replace ARG    Replacement String
 -o, --output-dir ARG Output Directory for edited files
Exiting ...
```

9.3.1 Analysis of the requirements

Processing the command line would involve handling a mutually exclusive set of switches **-s** and **-r**. In addition the switch **-R** is optional but should always be accompanied by **-o**. Fairly simple but typical of most command language driven utilities e.g. the C compiler.

Searching for a candidate string in a text file may be considered simple. Searching for a regular expression will need a library.

The input file is not overwritten and the substituted output shall be written to an output directory with the same name as the input file. File systems are the domain that this exposes us to.

In this projectlet **exception**s are used to announce unexpected situations - particularly deep in the implementation.

The implementation then depends on the predefined language environment:

```
~/bin/codemd ../../toolkit/examples/frep/src/impl.adb -x Environment -l
```

```
0002 | with Ada.Text_IO;       use Ada.Text_IO;
0003 | with Ada.Strings.Fixed; use Ada.Strings.Fixed;
0004 | with Ada.Directories;
0005 | with GNAT.Regpat;
0006 | with GNAT.Strings;
```

and for command line processing:

```
~/bin/codemd ../../toolkit/examples/frep/src/cli.adb -x Environment -l
```

```
0002 | with Ada.Text_IO; use Ada.Text_IO;
0003 |
0004 | with Ada.Directories;
0005 | with Ada.Exceptions;
0006 |
0007 | with GNAT.Command_Line;
0008 | with GNAT.Source_Info;
```

Packages referenced in the above, particularly **Ada.Exceptions**, **GNAT.Source_Info**, **GNAT.Regpat**, **GNAT.Strings** will be encountered repeatedly.

9.3.2 Command Line Processing

The workhorse for command line processing is the **GNAT.Command_Line** package. The switches that need to be handled are declared as:

```
~/bin/codemd ../../toolkit/examples/frep/src/cli.adb -x Define -l
```

```
0059 |       GNAT.Command_Line.Define_Switch
0060 |         (Config, Verbosity'Access, Switch => "-v:",
0061 |          Long_Switch => "--verbosity:", Help => "Verbosity Level");
0062 |
0063 |       GNAT.Command_Line.Define_Switch
0064 |         (Config, Candidate'Access, Switch => "-s:", Long_Switch => "--search:",
0065 |          Help                             => "Search Candidate");
```

```

      GNAT.Command_Line.Define_Switch
        (Config, CandidateExp'Access, Switch => "-r:",
         Long_Switch => "--regex:", Help => "Search Candidate - Reg Exp");

      GNAT.Command_Line.Define_Switch
        (Config, Replacement'Access, Switch => "-R:",
         Long_Switch => "--replace:", Help => "Replacement String");

      GNAT.Command_Line.Define_Switch
        (Config, outputdir'Access, Switch => "-o:",
         Long_Switch                      => "--output-dir:",
         Help => "Output Directory for edited files");

      GNAT.Command_Line.Getopt (Config, SwitchHandler'Access);
```

The command line is processed and checked for consistency and completeness before the application is initiated:

```
~/bin/codemd ../../toolkit/examples/frep/src/cli.adb -x Check -l
```

```
      if replacement.all'Length > 0 then
         Put ("Output Dir ");
         Put (outputdir.all);
         New_Line;
         if outputdir.all'Length < 1 then
            raise CLI_ERROR
              with "Please provide an output dir for edited files";
         end if;
         if not Ada.Directories.Exists (outputdir.all) then
            raise CLI_ERROR with "Non existent output dir";
         end if;
         if Ada.Directories.Kind (outputdir.all) /= Ada.Directories.Directory
         then
            raise CLI_ERROR with "Provide a directory for output";
         end if;
      end if;
```

Any inconsistent inputs are handled by utilizing the **exception** mechanism which immediately returns control to an exception handler or the enclosing environment if none provided.

9.3.3 Regular Expression handling

If the command line specified a regular expression either as a search candidate or as a replacement candidate, the same regular expression is applied to each file in the command line.

Optimal use of regular expressions involves ***compiling*** the expression once and use the compiled output repeatedly e.g. for each line in the input as illustrated:

```
~/bin/codemd ../../toolkit/examples/frep/src/impl.adb -x Compile -l
```

```
0096 |    patternStr : GNAT.Strings.String_Access;
0097 |    pcompiled  : access GNAT.RegPat.Pattern_Matcher;

...

0109 |       if patternStr = null then
0110 |          patternStr     := new String (candidate'Range);
0111 |          patternStr.all := candidate;
0112 |          pcompiled := new GNAT.RegPat.Pattern_Matcher'(Compile (candidate));
0113 |       end if;
```

Once compiled the matches of the expression in each line is straightforward:

```
~/bin/codemd ../../toolkit/examples/frep/src/impl.adb -x Search -l
```

```
0122 |       Open (file, In_FIle, filename);
0123 |       while not End_Of_File (file) loop
0124 |          Get_Line (file, line, linelength);
0125 |          linenumber := linenumber + 1;
0126 |          GNAT.RegPat.Match (pcompiled.all, line (1 .. linelength), matched);
0127 |          if matched (0) /= GNAT.RegPat.No_Match then
0128 |             Put (linenumber'Image);
0129 |             Set_Col (6);
0130 |             Put (" : ");
0131 |             Put (line (1 .. matched (0).First - 1));
0132 |             Put ("[");
0133 |             Put (line (matched (0).First .. matched (0).Last));
0134 |             Put ("]");
0135 |             Put (line (matched (0).Last + 1 .. linelength));
0136 |             New_Line;
0137 |             Count := Count + 1;
```

```
         end if;
      end loop;
      Close (file);
```

Of course if the candidate is a simple string, the compilation and the compiled expression are irrelevant but the general approach is identical.

If a replacement is requested, then instead of printing the results, the candidate is replaced by the replacement but written to a copy of the input file. Any lines which dont contain the candidate are copied verbatim to the output file:

```
~/bin/codemd ../../toolkit/examples/frep/src/impl.adb -x Replace -l
```

```
      Open (file, In_File, filename);
      Create (outfile, Out_File, outfilename);
      while not End_Of_File (file) loop
        Get_Line (file, line, linelength);
        GNAT.RegPat.Match (pcompiled.all, line (1 .. linelength), matched);
         if matched (0) = GNAT.RegPat.No_Match then
            Put_Line(outfile,line(1..linelength));
         else
            Put (outfile,line (1 .. matched (0).First - 1));
            Put (outfile,replacement);
            Put (outfile,line (matched (0).Last + 1 .. linelength));
            New_Line(outfile);
            Count := Count + 1;
         end if;
      end loop;
      Close (outfile);
      Close (file);
```

9.3.4 Exceptions

Handling and raising exceptions is a way to deal with anomalies particularly in deeply embedded code. Command Language processing utilizes this mechanism:

```
~/bin/codemd ../../toolkit/examples/frep/src/cli.adb -x Exception -l
```

```
   exception
      when GNAT.COMMAND_LINE.EXIT_FROM_COMMAND_LINE =>
        raise ;
```

```
0110 |       when e : others =>
0111 |          Put ("Exception ");
0112 |          Put (Ada.Exceptions.Exception_Name (e));
0113 |          Put (" ");
0114 |          Put (Ada.Exceptions.Exception_Message (e));
0115 |          New_Line;
0116 |          Put (usagestr);
0117 |          New_Line;
0118 |          raise;
```

in collaboration with the enclosing procedure:

```
~/bin/codemd ../../toolkit/examples/frep/src/frep.adb -x Exception -l
```

```
0050 |    exception
0051 |       when others =>
0052 |          Put_Line("Exiting ...");
```

9.3.5 Driving the entire project

Putting all of the above together, the utility analyzes the command line and calls the appropriate implementations to perform the required function:

```
~/bin/codemd ../../toolkit/examples/frep/src/frep.adb -x Overall -l
```

```
0009 |    cli.ProcessCommandLine;
0010 |    loop
0011 |       declare
0012 |          arg : constant String := cli.GetNextArgument;
0013 |       begin
0014 |          if arg'Length < 1 then
0015 |             exit;
0016 |          end if;
```

...

```
0028 |          if cli.Replacement.all'Length >= 1 then
0029 |             if cli.Candidate.all'Length >= 1 then
0030 |                impl.Replace
0031 |                  (arg, cli.Candidate.all, cli.Replacement.all,
                    cli.outputdir.all);
              elsif cli.CandidateExp.all'Length >= 1 then
                 impl.ReplaceRegEx
                   (arg, cli.CandidateExp.all, cli.Replacement.all,
                    cli.outputdir.all);
              end if;
           else
              if cli.Candidate.all'Length >= 1 then
                 impl.Search (arg, cli.Candidate.all);
              elsif cli.CandidateExp.all'Length >= 1 then
                 impl.SearchRegEx (arg, cli.CandidateExp.all);
              end if;
           end if;
        end;
     end loop;
```

9.4 Stretch

- Instead of introducing special markers in the output when candidates are found in the input, highlight the text in the output for the strings found. Optionally find all instances in each line instead of stopping at the first successful find. Optionally handle binary files instead of just text files.
- Code examples in the book have all been produced by the tool **codemd** which processes text files, looking for markup commands. Markup commands are specified and compiled as regular expressions somewhat patterned as above. The language agnostic implementation at https://gitlab.com/ada23/codemd.git can be improved e.g. to provide a caption and numbering for each segment of code.
- In the **GNAT** programming system, there is a third tool - a family of packages **GNAT.Spitbol** which provide arguably the most powerful pattern matching support. Based on the decades old **SNOBOL** pattern matching system, this set of packages bring the same power with an API. Implementing a preprocessor for C to convert the **enum** definitions to be more like **Ada** e.g. by adding support for attributes like ***Image***, ***Succ***, ***Pred*** is a worthwhile exercise.

9.5 Sample Implementation

```
Repository: toolkit
Direcotry: examples/vrex
Directory: examples/frep
```

10 Generics

In the earlier chapters we had brief encounters with containers like **Vector**s from the predefined library. Behind the curtains all the **container**s are **generic** in the sense that the individual items are of an unspecified data type. In **Ada** such generic packages though parameterized with the data type of the contained items, the entire source is compiled and verified for syntax and in many cases semantic consistency (where possible).

In this projectlet, we will tackle a challenge encountered in embedded applications by designing a generic package.

10.1 Problem Statement

Embedded applications tend to live ***forever*** ie once powered on, they function without any arbitrary terminations till explicitly shutdown in a planned way. An insulin delivery pump for example continues to perform its function thought it might ***sleep*** most of the time waking up periodically to deliver say a basal insulin dosage. Temperature control of houses, office buildings, robots operating in a warehouse assembling products are other examples.

In all such instances certain variables e.g. setpoints can be considered critical. Any corruption of such **critical** variables due to application malfunction, electrical malfunction or other such disturbances if not detected could lead to disastrous consequences.

Several techniques are followed in manipulating such critical variables:

- Protect with a checksum or a CRC. Recompute the checksum every time such a variable is updated and cross verify the CRC when accessing them.
- Store the values in multiple copies and compare them every time they are accessed. While accessing if the copies are not identical then handle this as a serious failure.

Both have their benefits; the first however may impose a moderately higher compute load which may be intolerable in an embedded environment. In this projectlet the latter approach is illustrated using Generics.

10.2 Design Consideration

The entire support should be designed to be agnostic to the type of the critical variable. In addition the constraint checking while retrieving the values would require specifial **Get** and **Set** support.

```
~/bin/codemd ../../toolkit/adalib/src/critical.ads -x Generic -l
```

```
generic

   type Item_Type is private;
   with function "=" (left, right : Item_Type) return Boolean;

package critical is

   AllocatorError     : exception;
   VariableCorruption : exception;

   type AllocatorType is
     access function (secondary : Boolean := False) return access Item_Type;
   procedure SetAllocators (a : not null AllocatorType);

   type Variable_Type is private;
   procedure Set (var : in out Variable_Type; value : Item_Type);
   function Get (var : Variable_Type) return Item_Type;

private
   type ItemPtr_Type is access all Item_Type;
   type Variable_Type is record
      primary   : ItemPtr_Type;
      secondary : ItemPtr_Type;
   end record;
end critical;
```

Of particular note is the **Allocator**. While typically the variables could just be anywhere in RAM, or heap, this package supports provide different allocators for the primary value and a distinct one for the backup or secondary copy. In embedded applications these might reside in distinct ***section***s relocatable at link time. This provides additional protection or ability to detect corruptions.

Of course once a corruption is detected, the consequence is dependent entirely on the application. This generic package simply raises an appropriate exception that can be handled by the application.

10.3 Getter and Setters for the variable

Values of the variables then have to be stored in distinct memory locations - provided by the allocator. While accessing the 2 copies are compared using the equality function ("=") which is a parameter to the generic package.

```
~/bin/codemd ../../toolkit/adalib/src/critical.adb -x GetSet -l
```

```
   procedure Set (var : in out Variable_Type; value : Item_Type) is
   begin
      if var.primary = null then
         if allocate = null then
            raise AllocatorError;
         end if;
         var.primary     := ItemPtr_Type (allocate.all);
         var.primary.all := value;

         var.secondary     := ItemPtr_Type (allocate.all (True));
         var.secondary.all := value;
         return;
      end if;

      if var.primary.all = var.secondary.all then
         var.primary.all   := value;
         var.secondary.all := value;
         return;
      end if;

      raise VariableCorruption;
   end Set;

   function Get (var : Variable_Type) return Item_Type is
   begin
      if var.primary.all = var.secondary.all then
         return var.primary.all;
      end if;
      raise VariableCorruption;
   end Get;
```

The first time a value is assigned, the allocators are invoked to assign the storage. Once allocated they are never deallocated which is typical for this application.

This implementation compares the 2 values while accessing as well as while the values are being adjusted.

10.4 Specialization

Depending on an application's critical variable, the generic package is **instantiate**d:

```
~/bin/codemd ../../toolkit/adalib/src/critical_float.ads -x Float -l
```

```
with critical;

package critical_float is new critical (Float, "=");
```

10.5 Application

This simple application has 2 critical variables which are manipulated as follows"

```
~/bin/codemd ../../toolkit/examples/cvar/src/cvars.ads -x Variables -l
```

```
with critical_float;
package cvars is
   tempSetpoint     : critical_float.Variable_Type;
   pressureSetpoint : critical_float.Variable_Type;
   function allocator (secondary : Boolean) return access Float;
end cvars;
```

The variables are defined in this package. This is not by necessity and is more a convenience. In particular the package body makes it automatic to provide the requisite allocators. In this example a trivial allocator is provided:

```
~/bin/codemd ../../toolkit/examples/cvar/src/cvars.adb -x Implement -l
```

```
package body cvars is
   function allocator (secondary : Boolean) return access Float is
   begin
      return new Float;
   end allocator;
begin
   critical_float.SetAllocators (allocator'Access);
end cvars;
```

The allocator which simply uses the **new** keyword which in turn allocates the memory from the default **heap** is provided. Initializing the allocator is done as part of the **elaboration** of the package. The elaboration happens implicitly without having to be explicitly invoked.

The variables are Set and retrieved using the appropriate methods:

```
~/bin/codemd ../../toolkit/examples/cvar/src/cvar.adb -x Application -l
```

```
with critical_float;
with cvars;

procedure Cvar is
begin
   critical_float.Set (cvars.tempSetpoint, 98.0);
   critical_float.Set (cvars.pressureSetpoint, 20.0);
   Put (critical_float.Get (cvars.tempSetpoint)'Image);
   Put (" ");
   Put (critical_float.Get (cvars.pressureSetpoint)'Image);
   New_Line;
end Cvar;
```

In this example, the critical variables are **Float**s. The critical variable may be of any datatype including **record**s.

10.6 Sample Implementation

```
Repository: toolkit
Direcotry: examples/cvar
```

11 Numerics

In this chapter, we will begin to shape our toolkit towards more specific scientific and engineering needs. As we survey the field, the gnu scientific library is a strong model to aspire to. In fact in a later chapter will build a binding directly to the library. Another family favored in the world of Python is the NumPy, SciPy and friends. In fact it wont be far fetched to assert these libraries have rendered Python a system of choice in machine learning, signal processing and other such applications.

The libraries are great but were they to be not easily adaptible to our needs, building one from scratch is a daunting effort. They tend to hide some of the practical issues and potential solutions e.g. comparison of floating point variables or dealing with zero divide issues and the like.

The projectlets of this chapter fill the engineer's toolkit with e.g. support for descriptive statistics.

11.1 Learning Objectives

- Real (floating point) arrays
- Basic descriptive statistics support.
- Random numbers and test data generation

11.2 Numbers Library for Floating Point data types

11.2.1 Core Definitions

Being a strongly typed language, we have to strategize be library development. **Ada** solution for this need is the **generic**s. In other words like the rest of the Predefined library, a generic package can be designed then instantiated for other data types. With a view to eventually transitioning in this direction, the core approach is to rename the standard data type **float** as **RealType** and use the latter throughout. When the package is migrated to be **generic** the specific data type (float, long_float or even Long_Long_Float) will be the instantiation parameter.

```
~/bin/codemd ../../toolkit/adalib/src/numlib.ads -x Core -l
```

```
package numlib is

   subtype RealType is Float;
   eps : constant RealType := 0.000_1;
   type RealArray is array (Integer range <>) of RealType;
   package RealVectors_Pkg is new Ada.Containers.Vectors (Natural, RealType);
   use RealVectors_Pkg;
   package Sorter_Pkg is new RealVectors_Pkg.Generic_Sorting ("<");

   function Convert (a : RealArray) return Vector;
   function Convert (v : Vector) return RealArray;

   function Create
     (length : Natural; default : RealType := RealType'First) return Vector;
   function Create (from : Vector) return Vector;
   function Create (length : Natural; low, high : RealType) return Vector;
   function Create (low, high, step : RealType) return Vector;

```

The above segment specifies several functions and procedures in terms of the data type RealType. Based on this a new **package** is created ie **RealVectors_Pkg** which affords a lot more power than the basic array supported by the language. E.g. This provides an **iteration** framework which we will be leveraging extensively.

However we also define some helpers **Convert** to ease the adoption of this package. In addition, specification and implementation of other support e.g. for pairwise operations is significantly improved.

```
~/bin/codemd ../../toolkit/adalib/src/numlib.ads -x Pair -l
```

```
   -- Pairwise arithmetic support
   procedure Add (v : in out Vector; value : Vector);
   procedure Sub (v : in out Vector; value : Vector);
   procedure Mult (v : in out Vector; value : Vector);
   procedure Div (v : in out Vector; value : Vector);

   procedure Append (v1 : in out Vector; v2 : Vector) renames
     RealVectors_Pkg.Append_Vector;
   function Append (v1 : Vector; v2 : Vector) return Vector;
```

11.2.2 Vector Creation

In the spirit of **NumPy** many flexible ways to create arrays:

```
~/bin/codemd ../../toolkit/adalib/src/numlib.adb -x Create -l
```

```
   function Create (from : Vector) return Vector is
      result : Vector;
   begin
      result := To_Vector (0.0, from.Length);
      Add (result, from);
      return result;
   end Create;

   function Create (length : Natural; low, high : RealType) return Vector is
      result : Vector             := Create (length);
      val    : RealType           := low;
      vdelta : constant RealType := (high - low) / Float (length - 1);
   begin
      for idx in 0 .. length - 1 loop
         result.Replace_Element (idx, val);
         val := val + vdelta;
      end loop;
      return result;
   end Create;
```

or a much more flexible, customizable appoach **:**

```
~/bin/codemd ../../toolkit/adalib/src/numlib.adb -x Populate -l
```

```
   function Create
     (length   : Natural;
      populate : not null access function
        (length, idx : Natural) return RealType)
      return Vector
   is
      use type Ada.Containers.Count_Type;
      result : Vector;
   begin
      result.Set_Length (Ada.Containers.Count_Type (length));
      for idx in 0 .. length - 1 loop
```

```
         Replace_Element (result, idx, populate (length, idx));
      end loop;
      return result;
   end Create;
```

This could be used for example to create signals to explore Digital Signal Processing algorithms.

11.3 Extensions - Statistics

In the following example, an array is created and populated with Normally distributed random variables:

```
~/bin/codemd ../../toolkit/adalib/src/numlib-statistics.adb -x Normal -l
```

```
   function Normal
     (length : Natural; mean : RealType := 0.0; stdev : RealType := 1.0)
      return Vector
   is
      result : Vector;
      procedure Normal (c : Cursor) is
         g : constant RealType := GNAT.Random_Numbers.Random_Gaussian (Gen);
      begin
         Replace_Element (result, To_Index (c), g * stdev + mean);
      end Normal;
   begin
      result.Set_Length (Ada.Containers.Count_Type (length));
      result.Iterate (Normal'Access);
      return result;
   end Normal;
```

11.3.1 Sample Runs

The following example generates 100 normally distributed random variates with a mean of 10 and a standard deviation of 5:

```
~/bin/codemd ../../toolkit/examples/stats/src/stats.adb -x Random -l
```

```
procedure random is
   use numlib.RealVectors_Pkg ;
   unit : constant String := Enclosing_Entity ;
   rv : Vector ;
begin
   if verbose
   then
      Put_Line(unit);
   end if;
   if Argument_Count > 1
   then
      rv := numlib.statistics.Normal(100,10.0,5.0);
      Put_Line("Normal ");
   else
      rv := numlib.statistics.Uniform(100);
      Put_Line("Uniform ");
   end if ;
   show(" " , rv);
end random ;
```

and a sample run of the above:

```
~/bin/stats r n
```

```
Stats.random
Normal
Vector
Length  100
Mean  1.04035E+01
Stdev  5.44411E+00
Values
 6.89268E+00, 1.34863E+01, 7.88176E+00, 9.10447E+00, 1.34546E+01, 1.19114E+01, 8.53111E+00,
 2.08609E+00, 1.93468E+01, 9.92501E+00, 1.08521E+01, 4.81616E+00, 7.66150E+00, 1.09800E+01,
 1.03812E+01, 7.74668E+00, 1.39057E+01, 1.33314E+00, 4.37127E+00,-1.46028E+00, 1.51027E+01,
 1.54312E+01, 1.28105E+01, 5.66142E+00, 7.96846E+00, 1.80275E+01, 2.00069E+01, 1.53938E+01,
 2.72482E+00, 7.31482E+00, 1.41641E+01, 6.43051E+00, 1.94287E+01, 8.58459E+00, 2.48774E+01,
 5.88362E+00, 1.26574E+01, 9.17087E+00, 3.03655E+00,-7.20230E-01, 3.58198E+00, 1.74868E+01,
 8.99496E+00, 1.13188E+01, 9.84350E+00, 9.32866E+00, 1.16671E+01, 8.52222E-01, 1.18996E+01,
 2.28308E+01, 2.09008E+01, 1.04644E+01, 1.28988E+01, 7.29402E+00, 9.65872E+00, 5.34604E+00,
 1.37380E+01, 1.55229E+01, 1.47116E+01, 1.12253E+01, 8.70695E+00, 1.52813E+01, 1.27516E+01,
 1.23011E+01, 1.05752E+01, 1.17954E+01, 7.80758E+00, 4.33020E+00, 1.14758E+01, 1.99570E+00,
 9.87057E+00, 1.39714E+01, 1.56334E+01, 3.81311E+00, 7.74414E+00, 4.02613E+00, 1.31363E+01,
 8.24636E+00, 9.04350E+00, 1.07669E+01, 5.43279E+00, 5.90409E+00,-2.54082E-01, 1.64418E+01,
 1.01381E+01, 1.44102E+01, 9.11493E+00, 1.20032E+01, 6.24473E+00, 1.91956E+01, 7.99577E+00,
 1.40970E+01, 1.36972E+01, 7.34773E+00, 8.97826E+00, 2.49183E+01, 1.04659E+00, 1.65986E+01,
 1.43930E+01, 1.26444E+01,
```

11.4 Extensions - Polynomials

With the toolkit chosen, polynomial evaluation and such become extremely simple:

```
~/bin/codemd ../../toolkit/adalib/src/numlib-polynomials.adb -x Eval -l
```

```
function Evaluate (v : Vector; x : RealType) return RealType is
   result : RealType := 0.0;
   procedure Evaluate (c : Cursor) is
   begin
      result := result * x + Element (c);
   end Evaluate;
begin
   v.Reverse_Iterate (Evaluate'Access);
   return result;
end Evaluate;
```

11.5 Stretch

- As developed now, the data items are of **Float** data type. Conversion to a generic approach to enable **Long_Float** and other **Fixed** data types would enhance the library. An example is https://github.com/RajaSrinivasan/numerics.git.
- Subsetting vectors using **window**s or **slice**s is a common need specifically for Signal processing applications.
- Higher order moments like skewness, kurtosis and other computations such as median and quantiles are extremely useful for data analysis.

11.6 Sample Implementation

```
Repository: toolkit
Repository: TechAdaBook
Directory: examples/stats
```

12 Semi Numericals

We now turn our attention away from numericals into semi-numerical applications. Well over 5 decades ago, ambitious engineers attempted to leverage the existing analog telephone network for digital transmission. With an acoustic coupler at each end, a dialup ***network*** could be established. The key to the success of this was ensuring the integrity of the data exchange; e.g. what you type at a terminal at one end had to be received at the computer exactly and vice versa. The solution that has evolved many generations started with **parity** checking at the individual character level, ***cyclic redundancy check*** at a block level and message digests of various flavors.

The problem of verifying the integrity of message content has not gone away.

A projectlet to compute and verify parity bits for characters will be our starting point. Further explorations with a 16 bit CRC computation and finally a message digest using the MD5 algorithm will be reviewed. For each technique, the ***payload*** will be corrupted and then ensured that the method does indeed provide different outputs.

The choice of the approach for any application is of course needs careful consideration of many factors. However all these approaches continue to be actively used and researched.

12.1 Learning Objectives

- Parity computations with low level bit manipulation
- Data Corruption simulation
- CRC computation
- Library exploration for message digests

12.2 Parity and Bit level manipulations

In a serial communications, computing a parity bit (Ref: https://en.wikipedia.org/wiki/Parity_bit) and transmitting it along with the data for comparing at the receiving point has been a standard solution from the early days. Instead of transmitting the 8 bits for each byte, the parity bit makes the payload 9 bits. Support for this is included in most serial port driver chips even now. In other words parity is handled mostly at the digital electronics level. Concepts will be explored in this projectlet for pedagogic purposes.

The basic specifications related to parity may be listed as:

```
~/bin/codemd ../../toolkit/adalib/src/parity.ads -x Basic -l
```

```
   type ParityType is (Odd, Even);

   type ParityBitType is (Low, High);

   type BitCountType is new Natural range 0 .. Unsigned_8'Machine_Size;
```

Though **odd** parity may be most common, other choice is **even** parity. Given a byte the parity bit computation of course depends on which is chosen:

```
~/bin/codemd ../../toolkit/adalib/src/parity.adb -x Parity -l
```

```
   odd_mask : constant := 2#0000_0001#;
   function ParityBit
     (byte : Unsigned_8; spec : ParityType := Odd) return ParityBitType
   is
      bc : constant Unsigned_8 := Unsigned_8 (BitCount (byte));
   begin
      if odd_mask = (bc and odd_mask) then
         case spec is
            when Odd =>
               return Low;
            when Even =>
               return High;
         end case;
      else
         case spec is
            when Odd =>
               return High;
            when Even =>
               return Low;
         end case;
      end if;
   end ParityBit;
```

As shown above, bitwise **and**, **or** etc are supported on **Unsigned** types; as are bit shift operations as shown below:

```
~/bin/codemd ../../toolkit/adalib/src/parity.adb -x Count -l
```

```
   function BitCount (byte : Unsigned_8) return BitCountType is
      result : Natural    := 0;
      mask   : Unsigned_8 := 1;
   begin
      for b in 1 .. 8 loop
         if mask = (mask and byte) then
            result := result + 1;
         end if;
         mask := Shift_Left (mask, 1);
      end loop;
      return BitCountType (result);
   end BitCount;
```

Even if basic parity computations are not implemented in software these days, the core features are useful in for example explicitly corrupting data blocks - for testing purposes:

```
~/bin/codemd ../../toolkit/examples/integ/src/integ.adb -x Corrupt -l
```

```
   function Corrupt (b : Unsigned_8) return Unsigned_8 is
      mask : Unsigned_8 := 0;
      rv   : Float ;
   begin
      loop
         rv := GNAT.Random_Numbers.Random (Gen);
         mask := Shift_Left (2#1#, Natural (rv * Float (mask'Size)));
         if mask /= 0
         then
            exit ;
         end if ;
      end loop ;
      return mask xor b;
   end Corrupt;
```

12.2.1 Sample Usage

What then is the result of corrupting 1 bit. In theory, the parity bit should then flip. With odd parity:

```
~/bin/integ p S r I n I
```

```
Argument : S 53
BitCount  4 Required Parity Bit HIGH
Corrupted to 73 Required Parity Bit LOW
Argument : r 72
BitCount  4 Required Parity Bit HIGH
Corrupted to f2 Required Parity Bit LOW
Argument : I 49
BitCount  3 Required Parity Bit LOW
Corrupted to 4b Required Parity Bit HIGH
Argument : n 6e
BitCount  5 Required Parity Bit LOW
Corrupted to 4e Required Parity Bit HIGH
Argument : I 49
BitCount  3 Required Parity Bit LOW
Corrupted to 09 Required Parity Bit HIGH
```

and with even parity:

```
~/bin/integ P I n I r S
```

```
Argument : I 49
BitCount  3 Required Parity Bit HIGH
Corrupted to c9 Required Parity Bit LOW
Argument : n 6e
BitCount  5 Required Parity Bit HIGH
Corrupted to 2e Required Parity Bit LOW
Argument : I 49
BitCount  3 Required Parity Bit HIGH
Corrupted to 09 Required Parity Bit LOW
Argument : r 72
BitCount  4 Required Parity Bit LOW
Corrupted to 52 Required Parity Bit HIGH
Argument : S 53
BitCount  4 Required Parity Bit LOW
Corrupted to d3 Required Parity Bit HIGH
```

12.3 CRC

Let us see how the CRC fares with 1 bit corruption:

```
~/bin/integ c "Hello World. This is a long sentence."
```

```
CRC is f861
With 1 bit corruption Corrupting char  11 (d) to '`'
CRC is 739f
With 1 bit corruption Corrupting char  4 (l) to 'n'
CRC is 6260
With 1 bit corruption Corrupting char  6 ( ) to '0'
CRC is 3869
With 1 bit corruption Corrupting char  25 (o) to 'k'
CRC is 7c6f
With 1 bit corruption Corrupting char  29 (s) to 'c'
CRC is 38ac
With 1 bit corruption Corrupting char  28 ( ) to '"'
CRC is 32c0
With 1 bit corruption Corrupting char  37 (.) to '&'
CRC is 3e60
With 1 bit corruption Corrupting char  37 (.) to '&'
CRC is 3e60
```

Except in rare cases - where the identical character is getting corrupted in an identical way, the resulting CRC should be different.

Of course with a 16 bit CRC, there are only 65535 (non 0) unique values and thus theoretically clashes in the final CRC cannot be ruled out. For practical applications, CRC can serve as a means to detect data corruptions.

12.4 Digest

Even stronger signatures with MD5 digests:

```
~/bin/integ h "Hello World. This is a long sentence."
```

```
Digest : db19e87d923af6dd2cfeb9a6fe71cbb4
With 1 bit corruption Corrupting char  35 (c) to '#'
```

```
Digest : e043ba7e4dec98c183c522045543eb89
With 1 bit corruption Corrupting char  7 (W) to ETB
Digest : 68d99e0470bb91c651f6e0a9973011d0
With 1 bit corruption Corrupting char  31 (n) to '.'
Digest : 9be7ff1e81ffa81b388bd83dc1bcc8f5
With 1 bit corruption Corrupting char  31 (n) to 'f'
Digest : 13ac19e797eb51823833cc3d17d457ca
With 1 bit corruption Corrupting char  7 (W) to '×'
Digest : de2ec0e0c919b95dcefd632985d6309a
With 1 bit corruption Corrupting char  32 (t) to 'T'
Digest : b0ff050b59e6b9451033331bd205067e
With 1 bit corruption Corrupting char  37 (.) to ','
Digest : 6a1b2c5538be67743e6cd047ac7e76b3
With 1 bit corruption Corrupting char  35 (c) to 'a'
Digest : fe5c8fe1522de8eb15ba39e78a120fd6
```

For CRC as well as the digests, all the results should result in a change.

12.5 Stretch

- Experiment with corruption of file contents and the use of CRC and/or digests to detect such corruptions. It is almost standard practice now for file servers to share the digital signatures to verify the downloaded versions are not currupt.
- How sensitive are the CRC and hash signatures to the order of characters in the input? Even if a corruption candidate and the corrupted value are identical, if the position within the target string is different, does the signature always change?
- The CRC used above is only 16 bits. Perhaps a 32 bit will provide even more security.
- Different popular tools like zip, use their specific versions of the CRC. Comparision of their performances might be highly educational.
- Widely used systems like **git** have chosen to adopt a higher order digital signature discipline over the MD5 algorithm. An analysis of file patterns that may result in MD5 clashes will be interesting.

12.6 Implementation

```
Repository: toolkit
Directory: examples/integ
```

13 Abstractions

In this chapter we tackle one of the heavier subjects related to application design ie **object orientation**. Another way to think about this is abstraction in terms of basic operations like storage, moving, conversion to different formats as well as behaviors. Ada supports such abstractions rather elegantly. In particular combined with generic's which enable a related form, gives developers a strong platform.

In two projectlets, we will tackle the many facets of abstraction.

13.0.1 Learning Objectives

- Operational abstraction
- Data Type abstraction
- Combination of generics and abstraction
- Data analysis and manipulation support tooling
- Introductory networking

13.1 Projectlet: Logging

The first projectlet is about data logging from applications. Such logs are generated at sub second rates in realtime applications while it may be at a lower rate in say a DevOps support utility. The key goal of such logging is of course to post process and analyze. More often than not, in a **system** logging from different sub modules need to be collated for system level analysis and diagnostics. In more extreme cases, the modules may be distributed across a network. Different aspects of such logging are abstracted leaving enough scope for further expansion.

13.1.1 Logging API

Essentially the logging needs of applications can be outlined in the form of an API as in:

```
~/bin/codemd ../../toolkit/adalib/src/logging.ads -x API -l
```

```
subtype message_level_type is Natural;

CRITICAL      : constant message_level_type := 10;
ERROR         : constant message_level_type := 20;
WARNING       : constant message_level_type := 30;
INFORMATIONAL : constant message_level_type := 40;
function Image (level : message_level_type) return String;

subtype Source_Name_Type is String (1 .. 6);
Default_Source_Name : Source_Name_Type := (others => '.');

subtype Message_Class_Type is String (1 .. 6);
Default_Message_Class : Message_Class_Type := (others => '.');

function Time_Stamp return String;

procedure SendMessage
  (message : String; level : message_level_type := INFORMATIONAL;
   source  : String := Default_Source_Name;
   class   : String := Default_Message_Class);
```

The application can provide a message in text in addition to some attributes like **severity,** the **source** or **function** for each message. Thus messages from different threads can be interspersed yet distinguished from each other.

The messages have to be **print**ed or somehow captured. The exact method of printing or collating is left unspecified or **abstract**ed giving us the flexibility of different implementations.

```
~/bin/codemd ../../toolkit/adalib/src/logging.ads -x Dest -l
```

```
type Destination_Type is abstract tagged record
   null;
end record;
procedure SetDestination (destination : access Destination_Type'Class);
procedure SendMessage
  (dest   : in out Destination_Type; message : String;
   level  :        message_level_type := INFORMATIONAL;
   source :        String             := Default_Source_Name;
   class  :        String             := Default_Message_Class) is abstract;
   procedure Close (desg : Destination_Type) is abstract;

   type StdOutDestination_Type is new Destination_Type with record
      null;
   end record;
```

In the simplest application, the logs just get sent to the ***Standard_Output*** channel. An application may then generate a trace as follows:

```
~/bin/codemd ../../toolkit/examples/logs/src/logs.adb -x Simple -l
```

```
   procedure T2 (argc : integer) is
      myname : constant String := GNAT.Source_Info.Enclosing_Entity;
   begin
      if argc = 0
      then
         logging.SetDestination (cstdout.handle'Access);
      end if ;
      Logging.SendMessage ("Message 1");
      Logging.SendMessage ("Critical ", logging.CRITICAL);
      Logging.SendMessage ("Error", logging.ERROR);
      logging.SendMessage ("Warning", logging.WARNING);
   end T2;
```

The above use case is the simplest with the application not needing any initialization with the logs going to **Standard Output**, resulting in:

```
~/bin/logs a
```

```
2025-02-06 10:25:16 ...... ...... [I] Message 1
2025-02-06 10:25:16 ...... ...... [C] Critical
2025-02-06 10:25:16 ...... ...... [E] Error
2025-02-06 10:25:16 ...... ...... [W] Warning
```

With a minor additional initialization, with no change to the application, the logs get sent to a different destination in this case a UDP socket (port no 1056 on the same host **localhost**) destination. Presumably a log server captures the logs via an UDP socket and thus able to collate logs from many sources:

```
~/bin/codemd ../../toolkit/examples/logs/src/socklog.adb -x Socklog -l
```

```
procedure T1 is
   myname  : constant String := GNAT.Source_Info.Enclosing_Entity;
   logsock : aliased logging.socket.SocketDestinationPtr_Type;
begin
   logsock := logging.socket.Create (1_056, "localhost");
   logging.SetDestination (logsock);
   for i in 1 .. 10 loop
      Logging.SendMessage ("Message 1");
      Logging.SendMessage ("Critical ", logging.CRITICAL);
      Logging.SendMessage ("Error", logging.ERROR);
      logging.SendMessage ("Warning", logging.WARNING);
      delay 2.0;
   end loop;
end T1;
```

13.1.2 Implementation

Internally of course different destinations require different handling. A socket destination requires a different initialization vs a text file destination. However, fundamental logic for creating a log is the same. This is achieved by designating an **abstract**, **tag**ged data type ie ***Destination_Type*** and at runtime the appropriate implementation for the specific destination is invoked by **dispatch**ing.

```
~/bin/codemd ../../toolkit/adalib/src/logging.adb -x Dispatch -l
```

```
procedure SendMessage
  (message : String; level : message_level_type := INFORMATIONAL;
   source  : String := Default_Source_Name;
   class   : String := Default_Message_Class)
is
begin
   SendMessage (Current_Destination.all, message, level, source, class);
end SendMessage;
```

and for the socket destination in particular:

```
~/bin/codemd ../../toolkit/adalib/src/logging-socket.adb -x SockDest -l
```

```
overriding procedure SendMessage
  (dest   : in out SocketDestination_Type; message : String;
      level  :       message_level_type := INFORMATIONAL;
      source :       String             := Default_Source_Name;
      class  :       String             := Default_Message_Class)
   is
      m       : aliased Message_Type;
      payload : Ada.Streams.Stream_Element_Array (1 .. m'Size / 8);
      for payload'Address use m'Address;
      last : Ada.Streams.Stream_Element_Offset;
   begin
      m.t   := Ada.Calendar.Clock;
      m.Seq := dest.Count + 1;
      m.l   := level;
      m.s   := source;
      m.c   := class;
      if message'Length > MAX_MESSAGE_SIZE then
         m.ml := MAX_MESSAGE_SIZE;
         m.mt := message (1 .. MAX_MESSAGE_SIZE);
      else
         m.ml                        := message'Length;
         m.mt (1 .. message'Length) := message;
      end if;
      GS.Send_Socket (dest.s, payload, last, To => dest.dest);
      if last /= m'Size / 8 then
         raise Program_Error with "Truncated datagram";
      end if;
      dest.Count := @ + 1;
   end SendMessage;
```

13.2 Projectlet: Tables

A totally different challenge occurs for example in the Data Analysis domain - analyzing huge amounts of related data most often in the form of **table**s. Most beautifully illustrated in the **tidyverse** and **dplyr** packages in the **R** framework. (Ref: https://tidyr.tidyverse.org. Similar support are available for **Julia** as well; thus the tools of choice in many big data applications particularly in ***Life Sciences/Healthcare***. The challenge in this case is the nature of data items - sometimes real, sometimes timestamps, other times ***categorical***. For a strongly typed language this is not very straightforward. The second projectlet of this chapter attempts to sketch a path to implementing such support in Ada.

```
~/bin/codemd ../../toolkit/adalib/src/tables.ads -x Abstract -l
```

```
   type ColumnType is abstract tagged record
      name : Unbounded_String;
   end record;

   procedure Append (col : in out ColumnType; value : String) is abstract;
   procedure Set
     (col : in out ColumnType; idx : Natural; value : String) is abstract;
   function Image
     (col : in out ColumnType; idx : Natural) return String is abstract;
   function Length (col : ColumnType) return Natural is abstract;
   procedure Remove (col : in out ColumnType; idx : Natural) is abstract;

   type ColPtrType is access all ColumnType'Class;
   --function Create (name : String) return ColPtrType is abstract;
   package TablePkg is new Ada.Containers.Vectors (Natural, ColPtrType);
   subtype TableType is TablePkg.Vector;
```

A table **TableType** is defined in the above to be a **Vector** for **Column**s. A **ColumnType** is an abstract data type with name a String being a common attribute for all its derivatives. Being a strongly typed language, any of the operations on elements of this column need to be data type specific. The **generic** support to the rescue:

```
~/bin/codemd ../../toolkit/adalib/src/tables.ads -x BasicTypes -l
```

```
   generic
      type T is private;
      with function Vfun (s : String) return T;
      with function Ifun (v : T) return String;
   package ColumnPkg is

      package ColumnValues_Pkg is new Ada.Containers.Vectors (Natural, T);
      subtype ColumnValuesType is ColumnValues_Pkg.Vector;

      type TColumnType is new ColumnType with record
         values : ColumnValuesType;
      end record;

      function Create (name : String) return ColPtrType;
      function Length (col : TColumnType) return Natural;
      procedure Remove (col : in out TColumnType; idx : Natural);
      function Get (col : TColumnType; idx : Natural) return T;

      procedure Append (col : in out TColumnType; value : String);
      procedure Set (col : in out TColumnType; idx : Natural; value : String);
      function Image (col : in out TColumnType; idx : Natural) return String;

   end ColumnPkg;
```

Unfortunately this solution does not work for all data types. For example, strings being of a varying size, need special handling. Similarly other data types such as **Date** and **Time** will require special handling. Overall architecture supports such specialized implementations:

```
~/bin/codemd ../../toolkit/adalib/src/tables.ads -x StrCol -l
```

```
   package StringColumnValues_Pkg is new Ada.Containers.Vectors
     (Natural, Unbounded_String);
   type StringColumnType is new ColumnType with record
      values : StringColumnValues_Pkg.Vector;
   end record;
```

where strings are converted to **Unbounded_String**s to be stored in such a table. The implementation for the abstract routines ***Append***, ***Image*** and so on perform the appropriate and necessary conversions:

```
~/bin/codemd ../../toolkit/adalib/src/tables.adb -x StrCol -l
```

```
   function Get (col : StringColumnType; idx : Natural) return String is
   begin
      return To_String (col.values.Element (idx));
   end Get;

   procedure Set (col : in out StringColumnType; idx : Natural; value : String)
   is
   begin
      StringColumnValues_Pkg.Replace_Element
        (col.values, idx, To_Unbounded_String (value));
   end Set;

   procedure Append (col : in out StringColumnType; value : String) is
   begin
      StringColumnValues_Pkg.Append (col.values, To_Unbounded_String (value));
   end Append;

   function Image (col : in out StringColumnType; idx : Natural) return String
   is
   begin
      return Get (col, idx);
   end Image;
```

13.2.1 Table Handling

Putting all of that together, the application needs to define the table:

```
~/bin/codemd ../../toolkit/examples/ptable/src/ptable.adb -x Table -l
```

```
   procedure T1 is
      tname         : constant String   := GNAT.Source_Info.Enclosing_Entity;

      atomic_number : tables.ColPtrType := integer_column.Create("AtomicNumber");
      symbol : tables.ColPtrType := tables.CreateStringColumn ("Symbol");
      name          : tables.ColPtrType := tables.CreateStringColumn ("Name");
      atomic_mass   : tables.ColPtrType := float_column.Create ("AtomicMass");
      pt            : tables.TableType;
   begin
      pt.Append (atomic_number);
      pt.Append (name);
      pt.Append (symbol);
      pt.Append (atomic_mass);

      tables.Load( Argument(1) , pt , "," );
      tables.Print(pt);
      tables.Save(Argument(1) & ".csv" , pt );
   end T1;
   -- CODEMD: END
begin
   if Argument_Count > 0 then
      T1;
   end if;
end Ptable;
```

and other operations to map the table to a CSV file are enabled:

```
~/bin/codemd ../../toolkit/adalib/src/tables-load.adb -x Load -l
```

```
procedure Load
  (filename : String; table : in out TableType; sep : String := ";")
is
   use type Ada.Containers.Count_Type;
   tblfile : Csv.File_Type;
begin
   tblfile := Csv.Open (filename, Separator => sep, FieldNames => False);
   loop
      for fld in 0 .. TablePkg.Length (table) - 1 loop
         declare
            --colname : String :=
            --  To_String (TablePkg.Element (table, Integer (fld)).name);
            colval  : constant String := Csv.Field (tblfile, Integer (fld) + 1);
         begin
            TablePkg.Element (table, Integer (fld)).Append (colval);
         end;
      end loop;
      if Csv.End_Of_File (tblfile) then
         exit;
      end if;
      Csv.Get_Line (tblfile);
   end loop;
   Csv.Close (tblfile);
end Load;
```

13.3 Stretch

- In order to handle the potential loss of log records, the network channel could be modified to use stream sockets and applications can take advantage with no change in the application.
- The file destination should be modified to create a new logfile every hour - totally transparent to the application.
- Datalogs with binary data instead of textual representation will be a really powerful tool to build time series data from realtime applications - particularly at high data rates.
- Additional data types for Date and Time will be needed.
- Support for all the **dplyr** features should be added to make this feature complete in this package.

13.4 Sample Implementation

```
Repository: toolkit
Directories: example/logs example/ptable
```

14 Concurrent Applications

In the last chapter **object orientation** was introduced as a means to build a rich logging support system. A particular example was logging over a **network** utilizing the **udp** discipline.

In this chapter, we build ourselves a server framework, supporting **stream** socket communications. A server framework that receives requests for services over a network channel would imply providing the services **concurrent**ly to many clients. A webserver is a typical server working in this fashion.

The server framework will also illustrate a far simpler form of object orientation supported by **Ada** in order to help realize diverse applications. A particular example will be a logging server - a marriage of the logging framework from the previous chapter and the server framework of this chapter; thus realizing a logging server that can receive log messages from many clients concurrently to present a uniform system log - invaluable in monitoring any application.

14.1 Learning Objectives

- Tasks and the concurrency framework
- Simple object orientation with **Interfaces**
- Stream Socket based client server implementations.

14.2 Simple interfaces

In this projectlet, a **function** with 1 argument returning a value is the subject of study. Such a function is defined as:

```
~/bin/codemd ../../toolkit/adalib/src/numlib.ads -x Spec -l
```

```
   type Fx is interface;
   type FxPtr is access all Fx'Class;

   function Val (F : Fx; x : RealType) return RealType is abstract;
```

and can be used as:

```
~/bin/codemd ../../toolkit/adalib/src/numlib-integration.ads -x Appl -l
```

```
   function Newton_Coates_3 (f : FxPtr; X1, X2 : RealType) return RealType;

   function Simpsons (f : FxPtr; X1, X2 : RealType) return RealType renames
     Newton_Coates_3;

   function Newton_Coates_4 (f : FxPtr; X1, X2 : RealType) return RealType;

   function Newton_Coates_5 (f : FxPtr; X1, X2 : RealType) return RealType;
```

Numerical Integration with 3 different algorithms (***Newton Coates***) are specified in the above with any arbitrary **function**. An application can provide the function as:

```
~/bin/codemd ../../toolkit/examples/diffint/src/funcs.ads -x Spec -l
```

```
   type PolySinusoid is new numlib.Fx with null record ;

   overriding
   function Val( f : PolySinusoid ; x : numlib.RealType ) return numlib.RealType ;

   tpf : aliased PolySinusoid ;
```

and its implementation:

```
~/bin/codemd ../../toolkit/examples/diffint/src/funcs.adb -x Body -l
```

and an application utilizes this as:

```
~/bin/codemd ../../toolkit/examples/diffint/src/diffint.adb -x Appl -l
```

```
      for n in 1..numSteps
      loop
         Put( x ); Put(" ; ");
         Put( funcs.Val(funcs.tpf , x )); Put(" ; ");
         y := y + numlib.integration.Newton_Coates_5(funcs.tpf'access , x , x + xdelta
         Put(y) ; Put(" ; ");
         New_Line ;
         x := x + xdelta ;
      end loop ;
```

At runtime, these produce CSV files of the computations:

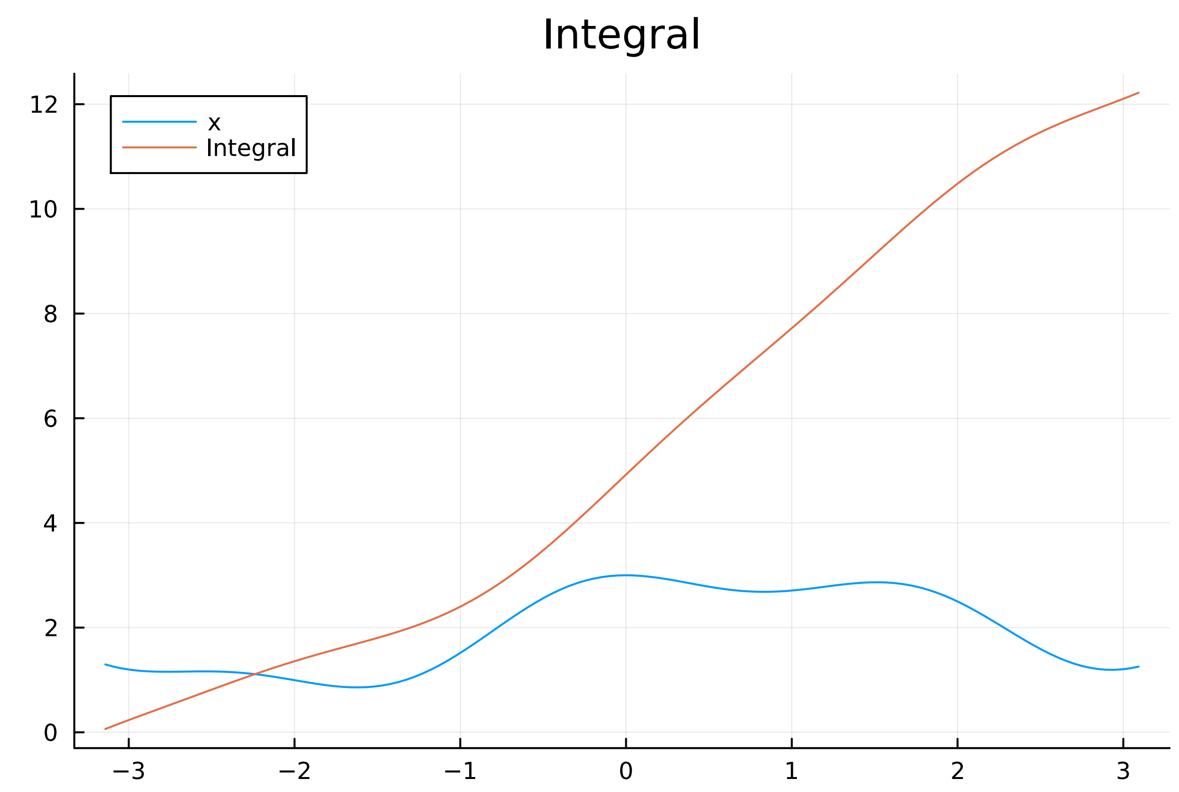
```
~/bin/diffint t3
~/bin/diffint t4
```

```
Diffint /Users/rajasrinivasan/bin/diffint Jan 13 2025 05:44:56
Diffint.T3
Diffint /Users/rajasrinivasan/bin/diffint Jan 13 2025 05:44:56
Diffint.T4
```

and plotting the output

```
using CSV
using DataFrames
using Plots

df=DataFrames.DataFrame(CSV.File("Diffint.T4.csv",delim=";",header=false));
plot(df.Column1 , [df.Column2,df.Column3] ,
                  label=["x" "Integral"] ,
                  title="Integral" , show=true)
```



The chart shows the original function and its integral.

14.3 Server framework

14.3.1 Communication Channel

In this projectlet Stream sockets are used as the communication channel. There are of course quite a few options and choices particularly for within a node but Stream based channels are the staple for distriputed systems. These channels provide reliable communication, bidirectional but have no clear markers between messages; in addition, the lower layers may break up a stream. Different standard tools/protocols such as **ftp**, **http** have used different techniques to address this challenge. Simple solutions such as a standard message size for all messages will work but will not scale very well.

In this projectlet, each message is preceded by a **length** field so that the receiver can mark the message boundaries correctly.

```
~/bin/codemd ../../toolkit/adalib/src/server.adb -x SendReceive -l
```

```
   procedure Send (sock : GS.Socket_Type; payload : AS.Stream_Element_Array) is
      hdr : constant HeaderType := HeaderType (payload'Length);
   begin
      HeaderType'Write (GS.Stream (sock), hdr);
      AS.Stream_Element_Array'Write (GS.Stream (sock), payload);
   end Send;

   function Receive (sock : GS.Socket_Type) return AS.Stream_Element_Array is
      hdr    : HeaderType;
      buffer : AS.Stream_Element_Array (1 .. MAX_MESSAGE_SIZE);
   begin
      HeaderType'Read (GS.Stream (sock), hdr);
      AS.Stream_Element_Array'Read
        (GS.Stream (sock), buffer (1 .. AS.Stream_Element_Offset (hdr)));
      return buffer (1 .. AS.Stream_Element_Offset (hdr));
   end Receive;
```

14.4 Multiple Clients

The quintessential server framework operates concurrently / independently or as a **task** in Ada terminology, with any other application, accepting requests for services from different ***client***s. For each client, a proxy is created also as an independent ***task*** that carries on a conversation with its client.

```
~/bin/codemd ../../toolkit/adalib/src/server.ads -x Server -l
```

```
   task type SockServer_Type is
      entry Serve (port : Integer; svc : ServicePtr_Type);
   end SockServer_Type;
   type SockServer_PtrType is access all SockServer_Type;

   -- proxy for each client
   task type ClientSock_Type is
      entry Serve
        (handler : ServicePtr_Type; sock : GS.Socket_Type;
         addr    : GS.Sock_Addr_Type);
   end ClientSock_Type;
   type ClientSockPtr_Type is access all ClientSock_Type;
```

In the above, the port identifies the service by number - used by a client to connect to the server and the argument **svc** the application the implements the service. The frame work expects an independent service to handle the initial connection request (procedure **ServiceConnection**) and subsequently every message received (procedure **Message**). Exactly how the message is handled is irrelevant to the server **task**.

The service itself is not too germane to the server framework except that it should support the handling of messages received from the client. Simply stated, the service is described as an **Interface** that meets some criteria:

```
~/bin/codemd ../../toolkit/adalib/src/server.ads -x Interface -l
```

```
type ServiceType is interface;
type ServicePtr_Type is access all ServiceType'Class;

-- Services that can be provided by a server
procedure ServiceConnection
  (svc  : in out ServiceType; Sock : GS.Socket_Type;
   From :        GS.Sock_Addr_Type) is null;
procedure Message
  (svc  : in out ServiceType; msgbytes : AS.Stream_Element_Array;
   Sock :        GS.Socket_Type; From : GS.Sock_Addr_Type) is null;
```

Thus the implementation relies on a singleton **task** to await connection requests:

```
~/bin/codemd ../../toolkit/adalib/src/server.adb -x Server -l
```

```
loop
   GS.Accept_Socket
     (mysocket, clientsocket, clientaddr, 5.0, Status => accstatus);
   if accstatus = GS.Completed then
      pragma Debug (Put_Line ("Received a connection"));
      sockclient := new ClientSock_Type;
      sockclient.Serve (handler, clientsocket, clientaddr);
   end if;
end loop;
```

As shown above, the server receives handles the connection request by creating a distinct **task** as a proxy for each client to handle the requests:

```
~/bin/codemd ../../toolkit/adalib/src/server.adb -x Clientproxy -l
```

```
      -- Let the handler do its thing
      myhandler.ServiceConnection (mysock, myaddr);
      loop
         HeaderType'Read (GS.Stream (mysock), msglen);
         AS.Stream_Element_Array'Read
           (GS.Stream (mysock),
            buffer (1 .. AS.Stream_Element_Offset (msglen)));
         myhandler.Message
           (buffer (1 .. AS.Stream_Element_Offset (msglen)), mysock, myaddr);
      end loop;
```

The proxy itself uses the **Interface** specification to invoke the service. The service is now very simple, just handling the message as required without having to worry about networking aspects.

14.5 Debug Server

A simple service that simply echoes the input message except converting the payload into a hexadecimal dump can be:

```
~/bin/codemd ../../toolkit/adalib/src/server.adb -x Debugserv -l
```

```
   procedure ServiceConnection
     (svc  : in out DebugService; Sock : GS.Socket_Type;
      From :        GS.Sock_Addr_Type)
   is
   begin
      Put (GSI.Source_Location);
      Put (" > New Connection from ");
      svc.cn := To_Unbounded_String (GS.Image (From));
      Put (To_String (svc.cn));
      New_Line;
   end ServiceConnection;

   procedure Message
     (svc  : in out DebugService; msgbytes : AS.Stream_Element_Array;
      Sock :        GS.Socket_Type; From : GS.Sock_Addr_Type)
   is
      resp : aliased constant String :=
        Hex.Image (msgbytes'Address, msgbytes'Length);
   begin
      Put (To_String (svc.cn));
      Put (" sent > Message of length ");
      Put (msgbytes'Length'Image);
      New_Line;
      Send (Sock, resp);
   end Message;
```

14.6 Clients of the framework

With the framework taking care of much of the network details, the clients can focus on the application. But first, the framework is wrapped into an executable with the wrapper code:

```
~/bin/codemd ../../toolkit/adalib/src/server.adb -x Debugserv -l
```

```
   procedure ServiceConnection
     (svc  : in out DebugService; Sock : GS.Socket_Type;
      From :        GS.Sock_Addr_Type)
   is
   begin
      Put (GSI.Source_Location);
      Put (" > New Connection from ");
      svc.cn := To_Unbounded_String (GS.Image (From));
      Put (To_String (svc.cn));
      New_Line;
   end ServiceConnection;

   procedure Message
     (svc  : in out DebugService; msgbytes : AS.Stream_Element_Array;
      Sock :        GS.Socket_Type; From : GS.Sock_Addr_Type)
   is
      resp : aliased constant String :=
        Hex.Image (msgbytes'Address, msgbytes'Length);
   begin
      Put (To_String (svc.cn));
      Put (" sent > Message of length ");
      Put (msgbytes'Length'Image);
      New_Line;
      Send (Sock, resp);
   end Message;
```

and a simple connection to this server can be made by a client as:

```
~/bin/codemd ../../toolkit/examples/clserv/src/clserv.adb -x DClient -l
```

```
procedure T2 is
   pl : Long_Float;
begin
   Put_Line (GSI.Enclosing_Entity);
   for i in 1 .. 20 loop
      pl := Long_Float (i);
      server.Send (sock, pl'Address, pl'Size / 8);
      declare
         resp  : AS.Stream_Element_Array := server.Receive (sock);
         resps : String (1 .. resp'Length);
         for resps'Address use resp'Address;
      begin
         Put (resps);
         New_Line;
      end;
   end loop;
end T2;
```

Which produces the following output when run:

```
~/bin/clserv t2
```

```
Clserv clserv.adb Jan 08 2025 22:40:37
Clserv.T2
000000000000f03f
0000000000000040
0000000000000840
0000000000001040
0000000000001440
0000000000001840
0000000000001c40
0000000000002040
0000000000002240
0000000000002440
0000000000002640
0000000000002840
0000000000002a40
```

```
0000000000002c40
0000000000002e40
0000000000003040
0000000000003140
0000000000003240
0000000000003340
0000000000003440
Clserv.T1
01000000
02000000
03000000
04000000
05000000
06000000
07000000
08000000
09000000
0a000000
```

the latter half being from a similar Integer exchange with the server. Note that both hex dumps indicate that the lowest order byte travels first which is a characteristic of ***little endian*** storage as opposed to ***big endian*** storage where the highest order octet would have traveled first.

14.7 Stretch

- The server framework inserts the payload length in the data stream; the main purpose being the ability to mark the message boundaries. Perhaps other conventions can be used such as in **http**. Modify the framework to enable building a webserver.
- Combine the logging framework and the server framework to realize a log server for a distributed application.
- Develop a file transfer utility similar to **ftp** utilizing the framework. This tool will use the same channel ie the stream to transfer the request and the payload ie the file itself - unlike the **ftp**.

14.8 Sample Implementation

```
Repository: toolkit
Directory: examples/clserv
Directory:examples/diffint
```

15 Libraries

So far, we have explored the predefined libraries extensively and have illustrated the applications of many. While the language specifications make it a rich and robust platform, building real life applications need more. All successful languages are backed by active and rich library system incorporating a combination of software as well as documentation. Recent examples include the ecosystems of **nodejs**, **golang**, **rust** and so on.

A recent addition - within the last few years - supporting **Ada** is the ada library system https://alire.ada.dev. The repository is a rich source of software support - in numerous disciplines; serving as components of applications as well as examples of good software engineering practises. Use of this library and the component crates is supported by **alr** a key driver of Ada software development.

In this chapter, we will attempt to incorporate some of these **crate**s into our own projects.

15.1 Learning Objectives

- Utilizing a library crate from **alire**
- Application revision management based on standard practises
- Tracking application binaries to sources

15.2 Projectlet fileres

Wordgames such as https://gitlab.com/ada23/words.git depend on a dictionary resource. Other applications frequently utilize audio files, graphics data as resources. This projectlet is designed to convert the resource files to a component of the executable image so that at runtime, there is no reason to reach into the file system of the host. Any file is translated into **Ada** source code which in turn is incorporated in the application executable. The applications at runtime may choose to use these resources directly in memory or by creating temporary files on the host system.

Resources such as dictionaries which are primarily textual can benefit a lot from ***compression***. This projectlet will then compress the file using the **zlib_ada** crate from **alire.ada.dev**. Since the data is compressed, at runtime the resource need to be ***decompress***ed before usage. The library supports both operations.

15.2.1 Getting started

Once the crate is chosen, incorporating it in a project is straightforward:

```
alr with zlib_ada
```

This directs alr to make appropriate modifications to add the zlib_ada to the project. The exact version of zlib_ada is recorded in the **alire.toml** file and the project config file is modified to include the reference to the crate:

```
head -5 ../../fileres/config/fileres_config.gpr
```

```
--  Configuration for fileres generated by Alire
with "zlib.gpr";
abstract project Fileres_Config is
   Crate_Version := "0.0.0";
   Crate_Name := "fileres";
```

The package then is used just like any other predefined library package:

```
~/bin/codemd ../../fileres/reslib/src/res-pack.adb -x Generate -l
```

```
0061 |          Ada.Streams.Stream_IO.Open
0062 |              (file,
0063 |              Ada.Streams.Stream_IO.In_File,
0064 |              name);
0065 |          stream := Ada.Streams.Stream_IO.Stream (file);
0066 |          declare
0067 |              buffer : Ada.Streams.Stream_Element_Array
0068 |                   (1 .. Ada.Streams.Stream_Element_Offset (filesize));
0069 |              bufferlen : Ada.Streams.Stream_Element_Offset;
0070 |          begin
0071 |              stream.Read (buffer, bufferlen);
0072 |              pragma Assert(Integer(bufferlen) = Integer(filesize) );
0073 |              Deflate_Init (Compressor);
0074 |              declare
```

```
        compressed_data : Ada.Streams.Stream_Element_Array
            (1 .. Ada.Streams.Stream_Element_Offset (filesize));
        compressed_size : Stream_Element_Offset;
    begin
        Translate (Compressor, buffer, I, Compressed_Data, compressed_size, Fi
        pragma Assert (I = Buffer'Last);
        Close (Compressor);

       Set_Output( resfile.file.all );
       New_Line ;
       Put_Line( Ascii.HT & Sanitize(Simple_Name(name)) & " : res.ResourceType
       Put_Line( Ascii.HT & Ascii.HT & "To_Unbounded_String( " & '"' & Simple_
       Put_Line( Ascii.HT & Ascii.HT & filesize'Image & " , ");
       Put_Line( Ascii.HT & Ascii.HT & "To_Unbounded_String (");
       Hex.dump.Dump
            (compressed_data'Address,
            Integer (bufferlen),
            false ,
            true ,
            32 ,
            resfile.file.all );

       Put_Line( Ascii.HT & Ascii.HT & '"' & '"' &  " )) ;") ;
       Set_Output( Standard_Output );
    end;
end ;
Ada.Streams.Stream_IO.Close (file);
```

As shown above, the file of interest is read verbatim, compressed using ***zlib*** and written to a source file for runtime access. For example, the file **res.ads** is rendered as follows:

```
~/bin/codemd ../../fileres/test/demo_resources.ads.keep -x Output -l
```

```
 RES_ADS : res.ResourceType := (
     To_Unbounded_String( "res.ads") ,
      221 ,
     To_Unbounded_String (
"789c758db10ec2300c44f77ec57d413f804c48cc0c2dccc8246e8928496527aa" &
"e0eb6994aa1378f2bbb3ef169f1e383a6afb243e8cda5ec33de6e0d8c1202bff" &
"f64c33937dd2c81056786d00a4f7cce8586316cb97025517b6515cd9d619fcc4" &
"815e8c03f6b05b8d87d96e7aff29fe9952169a76f94489febe71705bd1ca1514" &
"e60beb6946d6000090009e0401000000d06c426b01000000dd00000000000000" &
"104aa50401000000b06e426b01000000b06d4202080000000f00000008000000" &
"00000000ffffffff886e426b01000000104aa50401000000dc00a0040f" &
     "" )) ;
```

As shown above, the original filename and its raw size is included along with the compressed blob of data for runtime comparison:

```
~/bin/codemd ../../fileres/test/demo.adb -x Unpack -l
```

```
  declare
     d : Ada.Streams.Stream_Element_Array := res.unpack.Unpack( demo_resources.RES_P
     dstr : String( 1..d'Length ) ;
for dstr'Address use d'Address ;
  begin
     Put_Line(dstr);
  end ;
```

The unpacking operation utilizes the zlib:

```
~/bin/codemd ../../fileres/reslib/src/res-unpack.adb -x Unpack -l
```

```
Inflate_Init (Decompressor);
loop
   Translate
      (Decompressor,
         resvalstr
         (P + 1 ..
             Stream_Element_Offset'Min (P + Block_Size, L)) ,
         P,
         Uncompressed_Data
           (Total_Out (Decompressor) + 1 .. Uncompressed_Data'Last),
         O,
         No_Flush);
         if verbose
         then
         Ada.Text_IO.Put_Line
            ("Total in : " & zlib.Count'Image (Total_In (Decompressor)) &
               ", out : " & zlib.Count'Image (Total_Out (Decompressor)));
         end if ;
         exit when Total_Out (Decompressor) = L;
end loop;
```

At this point the application can utilize the decompressed data as needed either by creating a file or other processing. In the case of a dictionary, the data may be broken into individual words for inclusion in a more convenient, searchable data structure as:

```
~/bin/codemd ../../letters/boxed/src/dictionary.adb -x Dict -l
```

```
declare
   dictbytes : Stream_Element_Array := res.unpack.Unpack( fileres );
   dictstr : String( 1..dictbytes'Length );
      for dictstr'Address use dictbytes'Address ;
begin
   from := dictstr'First ;
   while from < dictstr'Last
   loop
      if Is_Letter(dictstr(from) )
      then
         for to in from+1..dictstr'Last
         loop
            if not Is_Letter( dictstr(to))
            then
               wordcount := wordcount + 1 ;
               Words_Pkg.Insert( result , To_Unbounded_String(dictstr(from..to-
               --Put_Line(dictstr(from..to-1));
               from := to ;
               exit ;
            end if ;
         end loop ;
      else
         from := from + 1 ;
      end if;
   end loop  ;
   Put("Dictionary contains "); Put(Integer(Words_Pkg.Length(result))); Put_Lin
end ;
```

15.3 Projectlet gitrev

The focus of this projectlet is to enable traceability from the binary to the source code. **git** being the predominant source code repository system, this projectlet attempts to capture enough tracking details to embed into the executable enabling querying and reporting of the exact set of sources that contributed to the binary. This leverages the **commitid** maintained by **git** regardless of the backend e.g. ***GitHub***, ***GitLab***, ***Bitbucket*** or whatever.

15.3.1 Git interaction

Using the **git** command line, we can obtain the relevant info. In this projectlet, we start by locating the executable:

```
~/bin/codemd ../../toolkit/adalib/src/git.adb -x Locate -l
```

```
   fullgit : constant GNAT.os_lib.String_Access := Locate_Exec_On_Path ("git");
```

For each query, the executable is invoked with appropriate command arguments in the appropriate directory (where the repository had been ***clone***d to then parse the output as required:

```
~/bin/codemd ../../toolkit/adalib/src/git.adb -x Exec -l
```

```
   -- Execute the command in the specified directory and return the output
   function Exec (dir : string; cmd : String) return String is
      cwd     : constant String := Ada.Directories.Current_Directory;
      status  : aliased Integer;
      arglist : constant Argument_List_Access := Argument_String_To_List (cmd);
   begin
      Ada.Directories.Set_Directory (dir);
      declare
         result : constant string :=
           GNAT.Expect.Get_Command_Output
             (fullgit.all, arglist.all, "", Status'Access, Err_To_Out => True);
      begin
         if Verbose then
            Put ("Dir: ");
            Put (dir);
            Put (" cmd: ");
            Put_Line (cmd);
            Put_Line (result);
         end if;
         Ada.Directories.Set_Directory (cwd);
         return result;
      end;
   exception
      when others =>
         Put ("Exception executing ");
         Put (cmd);
         Put (" @ dir ");
```

```
0093 |          Put_Line (dir);
0094 |          return "";
0095 |    end Exec;
```

While most of the time, the value printed is just what is needed, sometimes there are multiple lines of output eg. when branches are listed. We may have to process the output line before extracting the desired value by breaking the output into individual lines:

```
~/bin/codemd ../../toolkit/adalib/src/git.adb -x CurrentBranch -l
```

```
0169 |   function CurrentBranch (dir : String := ".") return String is
0170 |       cmd : String := "rev-parse --abbrev-ref HEAD" ;
0171 |    begin
0172 |       return Exec( dir , cmd );
0173 |    end CurrentBranch ;
```

on the other hand:

```
~/bin/codemd ../../toolkit/adalib/src/git.adb -x Tags -l
```

```
0244 |    function Tags (dir : String := ".") return wordlistpkg.Vector is
0245 |       taglines : constant String             := Exec (dir, "tag --list");
0246 |       result   : constant wordlistPkg.Vector := Get_Lines (taglines);
0247 |    begin
0248 |       return result;
0249 |    end Tags;
```

With this support generating a source file - just like **fileres** above that can be incorporated in the executable is an easy next step:

```
cat ../../toolkit/examples/gitrev/src/revisions.ads
```

```
--------------------------------------------
-- Created 2025-01-20 10:40:22
--------------------------------------------
package revisions is
    dir : constant String := "/Users/rajasrinivasan/Prj/GitLab/toolkit/examples/gitrev" ;
    version : constant String := "0.0.2" ;
    repo : constant String := "git@gitlab.com:ada23/toolkit.git" ;
    commitid : constant String := "bbc1d79bdac1eacd0c3fd7cac22dd01022eff956" ;
    abbrev_commitid : constant String := "bbc1d79" ;
    branch : constant String := "main" ;
end revisions ;
```

15.3.2 Version Tracking

Another key component is version tracking and the standard to adopt is https://semver.org/. **alire** of course provides appropriate support which can be incorporated in an application similar to ***zlib_ada*** above:

```
alr with semantic_versioning
```

The application **gitrev** makes a very light use of semantic versioning - insisting that the user provided revision string conforms to the standard format. Internally gitrev itself attempts to use semantic versioning:

```
~/bin/codemd ../../toolkit/examples/gitrev/src/gitrev.adb -x semver -l
```

```
   declare
      vstring : constant String :=
               Semantic_Versioning.Image(
               Semantic_Versioning.Parse( cli.version.all ) ) ;
      argdir       : constant String := To_String (dir);
      specfilename : constant String := cli.outputFile.all;
      specfile     : File_Type;
   begin
      Create (specfile, Out_File, specfilename & ".ads");
      Set_Output (specfile);
      Put_Line (longcomment);
      Put (comment);
      Put ("Created ");
      Put_Line (Local_Image (Clock));
      Put_Line (longcomment);
      Put ("package ");
      Put (specfilename);
      Put_Line (" is");
      StringConstOutput ("dir", Full_Name (argdir));
      New_Line;
      StringConstOutput ("version", vstring );
      New_Line;
```

and at runtime:

```
~/bin/gitrev -v
```

```
Command Line Arguments
Verbose TRUE
Output File revisions
Version 0.0.1
gitrev version
Repo git@gitlab.com:ada23/toolkit.git
Version 0.0.2
Commit Id bbc1d79bdac1eacd0c3fd7cac22dd01022eff956
Please provide a dir. Use '.' for current
```

15.4 Stretch

- fileres is very much **Ada** centric keeping up with the spirit of this book. Resources are of course need in applications regardless of programming language. Extending this to a scripting language such as **python** may be quite interesting.
- There are more featureful crates in the **alire** repository to perform such resource embedding. fileres could be extended to achieve feature parity.
- gitrev and supporting libraries potentially can be extended to build an automatic build system like https://www.jenkins.io.
- An automatic build system that generates build numbers for the semantic versioning scheme outlined above will be quite useful.

15.5 Sample Implementation

```
Repository: fileres
Repository: toolkit
Directory: example/gitrev
```

16 The wide world

The ecosystem of **Ada** development centered around **alire** has vastly expanded the free resources available for learning or for practical use in our own applications. Some of the ***crate***s like https://alire.ada.dev/crates/gsl provide a conduit to full featured, open source libraries like https://www.gnu.org/software/gsl/ that are not in ada by building bindings. While it is definitely feasible to build such a library entirely from scratch in **Ada**, the bindings provide continued access to the brilliant minds that contribute to such libraries.

The subject of this projeclet is https://mosquitto.org; in particular applications utilizing the **MQTT** protocol. Starting from a **thin** binding to the mosquitto library which is automatically generated, a layer of **Ada** like interfaces (or packages) are developed. A simple application is developed to illustrate the features.

16.1 Learning Objectives

- Steps involved in building bindings
- Thin vs Thick or Thickish bindings
- Communication patterns beyond client/server
- Leveraging testing resources

16.2 Numerical Expression Captcha

The goal of this projectlet is to support https://en.wikipedia.org/wiki/CAPTCHA except the challenge will be numerical expressions expecting the responder to compute the answer ***prove*** that they are not an automaton. Since the numerical expressions are dynamically and randomly generated the answers are not static and thus provides a reasonable challenge.

16.2.1 High Level design

The support is separated into a quizzer which is responsible for challenging the user and providing the necessary permissions. The quizzer however does not generate the numerical expression - relying on a poser of quizzes to provide a problem as well as the solution. Such a communication utilizes the **MQTT** protocol, in particular the **mosquitto** library opening up many possibilities.

In this projectlet, the communication patterns utilized is very simple - periodic broadcast. Every minute or so the poser broadcasts an automatically generated puzzle and its solution. The quizzer listens for such a broadcast for challenging the user till a successful solution is provided by the user.

16.2.2 Binding generation

The first step in this effort is to generate a **thin** binding to the library - which translates the basic definitions from presumably the native **C** specifications to **Ada**.

```
g++ -fdump-ada-spec /opt/homebrew/include/mosquitto.h
```

On a MacOS, for example the following generates the basics :

```
arm_types_h.ads arm_utypes_h.ads mosquitto_h.ads stddef_h.ads stdint_h.ads sys_upthread_upthi
```

Of these package specifications, only *__mosquitto_h.ads__ is specific to the library of interest while others all are the specifications of the package.

```
head -25 ../../bindings/adamosquitto/work/mosquitto_h.ads
```

```
pragma Ada_2012;

pragma Style_Checks (Off);
pragma Warnings (Off, "-gnatwu");

with Interfaces.C; use Interfaces.C;
with Interfaces.C.Strings;
with System;
with Interfaces.C.Extensions;
with stddef_h;
with utypes_uuint8_t_h;
with utypes_uuint16_t_h;
with utypes_uuint32_t_h;

package mosquitto_h is

   LIBMOSQUITTO_MAJOR : constant := 2;  --  /opt/homebrew/include/mosquitto.h:67
   LIBMOSQUITTO_MINOR : constant := 0;  --  /opt/homebrew/include/mosquitto.h:68
   LIBMOSQUITTO_REVISION : constant := 20;  --  /opt/homebrew/include/mosquitto.h:69
   --  unsupported macro: LIBMOSQUITTO_VERSION_NUMBER (LIBMOSQUITTO_MAJOR*1000000+LIBMOSQUIT

   MOSQ_LOG_NONE : constant := 0;  --  /opt/homebrew/include/mosquitto.h:74
   MOSQ_LOG_INFO : constant := (2**0);  --  /opt/homebrew/include/mosquitto.h:75
   MOSQ_LOG_NOTICE : constant := (2**1);  --  /opt/homebrew/include/mosquitto.h:76
   MOSQ_LOG_WARNING : constant := (2**2);  --  /opt/homebrew/include/mosquitto.h:77
```

16.2.3 Prettier Package

Beginning with renaming the package from mosquitto_h to mosquitto, some simple heuristics provide a prettier interface that renders the users more readable:

```
head -25 ../../bindings/adamosquitto/src/mosquitto.ads
```

```
pragma Ada_2012;

pragma Style_Checks (Off);
pragma Warnings (Off, "-gnatwu");

with Interfaces;   use Interfaces;
with Interfaces.C; use Interfaces.C;
with Interfaces.C.Strings;
with System;
with Interfaces.C.Extensions;

package mosquitto is

   LIBMOSQUITTO_MAJOR    : constant :=
     2;  --  /opt/homebrew/include/mosquitto.h:67
   LIBMOSQUITTO_MINOR    : constant :=
     0;  --  /opt/homebrew/include/mosquitto.h:68
   LIBMOSQUITTO_REVISION : constant :=
     20;  --  /opt/homebrew/include/mosquitto.h:69
   --  unsupported macro: LIBMOSQUITTO_VERSION_NUMBER (LIBMOSQUITTO_MAJOR*1000000+LIBMOSQUITT

   MOSQ_LOG_NONE : constant := 0;  --  /opt/homebrew/include/mosquitto.h:74
   MOSQ_LOG_INFO        : constant :=
     (2**0);  --  /opt/homebrew/include/mosquitto.h:75
   MOSQ_LOG_NOTICE      : constant :=
```

As shown below, the macros for error codes have been shortened to remove redundant and repetitive phrases:

```
~/bin/codemd ../../bindings/adamosquitto/src/mosquitto.ads -x Errors -l
```

```
   subtype mosq_err_t is int;
   AUTH_CONTINUE          : constant mosq_err_t := -4;
   NO_SUBSCRIBERS         : constant mosq_err_t := -3;
   SUB_EXISTS             : constant mosq_err_t := -2;
   CONN_PENDING           : constant mosq_err_t := -1;
   SUCCESS                : constant mosq_err_t := 0;
   NOMEM                  : constant mosq_err_t := 1;
   PROTOCOL               : constant mosq_err_t := 2;
   INVAL                  : constant mosq_err_t := 3;
   NO_CONN                : constant mosq_err_t := 4;
   CONN_REFUSED           : constant mosq_err_t := 5;
   NOT_FOUND              : constant mosq_err_t := 6;
   CONN_LOST              : constant mosq_err_t := 7;
   TLS                    : constant mosq_err_t := 8;
   PAYLOAD_SIZE           : constant mosq_err_t := 9;
```

The functions (and procedures) have been renamed in the specification though the **Import** pragmas provide the complete **C** external name.

```
~/bin/codemd ../../bindings/adamosquitto/src/mosquitto.ads -x Functions -l
```

```
   function alloc
     (id  : Interfaces.C.Strings.chars_ptr; clean_session : Extensions.bool;
      obj : System.Address)
      return Handle  -- /opt/homebrew/include/mosquitto.h:314
   with
     Import => True, Convention => C, External_Name => "mosquitto_new";

   procedure destroy
     (mosq : Handle)  -- /opt/homebrew/include/mosquitto.h:327
   with
     Import => True, Convention => C, External_Name => "mosquitto_destroy";
```

16.2.4 Thickish Layer

Though the above package is functional and could support most applications, it is not **Ada** friendly. For example all strings are expected to be null terminated as is the norm in the **C** world. We help ourselves by building a family of packages which provide a lot more **Ada**ish interfaces.

```
~/bin/codemd ../../bindings/adamosquitto/src/mqtt.ads -x Adaish -l
```

```
package mqtt is

   verbose : Boolean := True;

   INITIALIZATION_FAILURE  : exception;
   HANDLE_CREATION_FAILURE : exception;
   CONNECT_FAILURE         : exception;

   test_server           : constant String  := "test.mosquitto.org";
   -- These use cleartext messages
   test_server_port : constant Integer := 1_883;       -- No Username, password
   test_server_port_auth : constant Integer := 1_884;  -- Authenticated. username+pas

   function ErrorMessage (code : mosquitto.mosq_err_t) return String;

   function Create (id : String := "") return mosquitto.Handle;
   procedure Destroy (h : mosquitto.Handle ) renames mosquitto.destroy;
```

The burden of handling strings is eliminated with this layer. Similarly status codes are converted into **Exception**s.

Further, the interfaces required for a **producer** being distinct from those of the **consumer** of the information, individual packages provide a friendlier interface:

16.2.4.1 Producer

```
~/bin/codemd ../../bindings/adamosquitto/src/mqtt-publisher.ads -x Publisher -l
```

```
package mqtt.Publisher is

   procedure Connect
     (h        : mosquitto.Handle; server : String := test_server;
      port : Integer := test_server_port; will_topic : access String := null;
      will_msg : access String := null; keepalive : Integer := 60);

   procedure Start (h : mosquitto.Handle);

   procedure Publish
     (h   : mosquitto.Handle; topic : String; payload : String;
      qos : Integer := 2; retain : Boolean := False);

end mqtt.Publisher;
```

16.2.4.2 Consumer

```
~/bin/codemd ../../bindings/adamosquitto/src/mqtt-subscriber.ads -x Subscriber -l
```

```
package mqtt.subscriber is
   SUBSCRIBE_ERROR : exception;
   type MessageHandler is
     access procedure (msg : access constant mosquitto.mosquitto_message);

   procedure Connect
     (h     : mosquitto.Handle; msghandler : not null MessageHandler;
      topic : String; server : String := test_server;
      port  : Integer := test_server_port; keepalive : Integer := 60);
   procedure Start (h : mosquitto.Handle);
end mqtt.subscriber;
```

16.3 Projectlet Quiz

With the layers of support outlined above, a producer of the quiz or ***publisher*** in **MQTT** terminology becomes simple:

```
~/bin/codemd ../../bindings/adamosquitto/examples/pquiz/src/pquiz.adb -x Publisher -l
```

```
   h := mqtt.Create ;
   mqtt.Publisher.Connect(h) ;
   mqtt.Publisher.Start(h);

   for i in 1..10
   loop
      delay 60.0;
      pexpr := numexpr.Generate(2);
      pubtime := Ada.Calendar.Clock ;
      declare
         payload : String := Ada.Calendar.Formatting.Local_Image(pubtime) & "::" &
                                         numexpr.Image(pexpr) &
                                         " == " &
                                         numexpr.Image( numexpr.Evaluate(pexpr) );
      begin
         mqtt.Publisher.Publish(h,"pquiz" , payload);
         Put_Line(payload);
      end ;
   end loop ;
```

and the subscriber with a little help from an external package to handle each message received:

```
~/bin/codemd ../../bindings/adamosquitto/examples/quizzer/src/quizzer.adb -x Quizzer -l
```

```
procedure Quizzer is
   h : mosquitto.handle;
begin
   h := mqtt.Create;

   mqtt.subscriber.Connect(h,
                           handlers.strPayload'Access,
                           "pquiz");
   mqtt.subscriber.Start(h);

   for i in 1..10
   loop
      Put_Line("Waiting");
```

```
          delay 60.0 ;
       end loop ;
    end Quizzer;
```

16.4 Stretch

- Instead of a periodic broadcast, convert the interaction to a query/response interaction which is more conventional
- Introduce additional **topic**s in **MQTT** parlance - providing different types of quizzes - word puzzles for instance
- The test servers used in the examples are meant for testing purposes. Setup the **broker**s dedicated to this application.

16.5 Sample Implementation

```
Repository: adamosquitto
Directory: examples/pquiz
Directory: examples/quizzer
Directory: docker
```

17 Summary

Time has come to wrap up this effort, this labor of passion. Over 2 dozen chapters, the book presented well over 2 dozen **Projectlet**s starting from introductory file input (**fileio**), stretching to a full featured applications (**codemd**, **gitrev** and the **quizzer** framework). Numerous techniques were explored with special emphasis on utilizing the predefined language environment. Some of the projectlets leveraged the **ecosystem** of **Ada** while reaching out for special needs - **Julia** for graphing, **C** for standard libraries and so on.

In addition to being able to access the codebase, the student/reader can tap into the **Docker** support whereby a container based on **Ubuntu** linux is enabled along with the build of all the applications referenced. Regardless of your native development environment Windows or MacOS or Linux, the containers enable you to experiment. This is a particularly beneficial support tool that should encourage you to tackle the **Stretch** projects outlined.

By no means is this work complete. Pragmatics of problem solving tools and techniques have been the focus of this book. The fundamentals of syntax and semantics have been glossed over, preferring instead to refer to works of such giants as:

Norman Cohen, Michael Feldman and John Barnes.

Companion works focussing on **Provably correct applications using Spark**, **Embedded Real Time** applications, **Digital Signal Processing**, and **User Interface**s are under development; translations of this to other programming languages like **Swift** and **Go** are also in the roadmap - albeit in a distant future.

Feedback is always welcome by email to me.

A Code Markdown

Bulk of this book is code snippets extracted from source files. Instead of entire files, sections of sources are extracted but printed with line numbers so that they could be traced in the original sources. The ***snippet***s solve specific problems in implementation or illustrate a technique or otherwise are interesting.

The snippet extraction and printout is performed by a tool **codemd**. With a simple markup language embedded in source code, the tool prepares source code for an expository presentation.

A.1 Learning Objectives

- Another pattern matching application

A.2 Snippet Extraction

A.2.1 Code Markdown

Inspired by tools such as Doxygen and of late, the more ubiquitous markdown approaches, this tool supports few keywords:

```
codemd: begin segment=<segment name> caption=<A caption>
codemd: end
```

The above identifies a segment of code to extract and printout along with line numbers.

```
codemd: skipbegin
codemd: skipend
```

This pair enables some section embedded within the basic begin/end pair to be skipped. Typically uninteresting parts of a lengthy segment are skipped to keep the output somewhat manageable.

A.2.2 Patterns of interest

The above specifications are translated into regular patterns using the package GNAT.Regpat as follows:

```
~/bin/codemd ../../codemd/src/impl.adb -x patterns -l
```

```
0015 |    cmdid      : constant String          := "codemd" & ":";
0016 |    beginpat   : constant Pattern_Matcher :=
0017 |      Compile
0018 |        (cmdid & " +begin" & " +segment=([a-zA-Z0-9]+)" &
0019 |         " +caption=([a-zA-Z0-9 \-]+)");
0020 |    endpat     : constant Pattern_Matcher := Compile (cmdid & " +end");
0021 |    skippat    : constant Pattern_Matcher := Compile (cmdid & " +skipbegin");
0022 |    skipendpat : constant Pattern_Matcher := Compile (cmdid & " +skipend");
```

Each line of the source file is scanned and pattern matched to direct the actions:

```
~/bin/codemd ../../codemd/src/impl.adb -x Process -l
```

```
0093 |       Put_Line(inputfilename);
0094 |       Open (inpfile, In_File, inputfilename);
0095 |       while not End_Of_File (inpfile) loop
0096 |          Get_Line (inpfile, inpline, inplinelen);
0097 |          lineno := lineno + 1;

...

0110 |       end loop;
0111 |       Close (inpfile);
```

As seen above, the markdown directives can be embedded in any language source file. All languages provide a way to separate comment lines from code which is leveraged by **codemd**.

A.3 Stretch

- More language aware output e,g. keywords in bold
- Ability to render the snippets prettier

A.4 Sample Implementation

```
Repository: codemd
```

B Docker Container

This book and others which target hands-on developers contain numerous code examples. **Container** technology (https://en.wikipedia.org/wiki/Open_Container_Initiative) is an effective tool to distribute the corpus of software libraries and examples to enable students to replicate the results on their own infrastructure. In order to experiment with e.g. **MQTT** support outlined elsewhere in this book, a container with all the prerequisites can be and is made available.

In fact a container is made available that installs, builds the entire **toolkit** discussed extensively in this book. Also included are all the examples.

B.1 Learning Objectives

- Replicating all the examples in this book

B.2 Toolkit Container

The only prerequisites for the toolkit are the compilation system and the library manager **alire**. Assuming the compiler system is installed the following specifications performs the basic preparatory steps:

```
~/bin/codemd ../../toolkit/examples/containers/docker/Dockerfile -x AlireBuild -l
```

```
WORKDIR /projects
RUN git clone --recurse-submodules https://github.com/alire-project/alire.git
RUN cd alire; dev/build.sh ; cp bin/alr /bin
RUN alr toolchain --disable-assistant
```

Once alire is build and made available, the library and all the examples are built:

```
~/bin/codemd ../../toolkit/examples/containers/docker/Dockerfile -x Toolkit -l
```

```
RUN git clone https://gitlab.com/ada23/codemd.git
RUN cd codemd; alr build; cp bin/codemd /usr/local/toolkit/bin
RUN git clone https://gitlab.com/ada23/toolkit.git
RUN cd toolkit/adalib; alr build
RUN cd toolkit/examples/;  ./build.sh; cp bin/* /usr/local/toolkit/bin

RUN git clone https://gitlab.com/RajaSrinivasan/TechAdaBook.git
RUN cd TechAdaBook/Prj; ./build.sh; cp bin/* /usr/local/toolkit/bin
ENV PATH /usr/local/toolkit/bin:$PATH
```

The container image can then be produced resulting in an **Ubuntu** based image:

```
docker build --no-cache -t toolkit:latest .
```

The image can now be accessed:

```
docker run -it toolkit:latest bash
```

thus enabling access.

B.3 Stretch

- Evaluate and experiment with other container tools such as https://appimage.org and https://www.vagrantup.com

B.4 Sample Implementation

```
Repository: toolkit
Directory: examples/containers/docker
Repository: adamosquitto
Directory: examples/docker
```